SAVONIA

Siltojen kunnonvalvonta

Ville Pyykölä

Opinnäytetyö

___. ___. _____ _____

Ammattikorkeakoulututkinto

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ**
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Elektroniikan koulutusohjelma	
Työn tekijä(t) Ville Pyykölä	
Työn nimi Siltojen kunnonvalvonta	
Päiväys 10.1.2013	Sivumäärä/Liitteet 70+40
Ohjaaja(t) pt. tuntiopettaja Seppo Karjalainen	
Toimeksiantaja/Yhteistyökumppani(t) Savonia-ammattikorkeakoulu	

Tiivistelmä

Opinnäytetyön tarkoituksena oli toteuttaa laitteisto, jolla voitaisiin mitata voimaa ja värähtelyä sekä lähettää mitattu data langattomasti tietokoneelle. Lisäksi tarkoituksena oli tutustua siltojen kunnonvalvontajärjestelmiin. Työ tehtiin Savonia-ammattikorkeakoululle Kuopioon.

Siltojen kunnonvalvontaan liittyen toteutettiin mittausjärjestelmä, jossa on mukana datankeräysominaisuus. Mittausjärjestelmää varten opiskeltiin venymäliuskojen ja kiihtyvyysanturin käyttö. Venymäliuskoilla mitataan voimaa ja kiihtyvyysanturilla värähtelyä. Työssä käytettiin venymäliuskamittauksissa Acam Picostrain PS081 -venymäliuskakontrolleria. Värähtelynmittauksessa sekä langattomassa datan siirrossa käytettiin Texas Instrumentsin MSP430-mikrokontrolleria. Molempien mikrokontrollerien käyttö ja ohjelmointi opiskeltiin ja otettiin käyttöön tarvittavat ohjelmistot. Datankeräysjärjestelmää varten tehtiin sovellus, joka kerää mitattua dataa tietokoneelle.

Lopputuloksena saatiin toimiva laite, joka mittaa värähtelyä ja lähettää datan langattomasti tietokoneelle.

Avainsanat
voiman mittaus, värähtelyn mittaus

Julkinen

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Electronic Engineering			
Author(s) Ville Pyykölä			
Title of Thesis Condition Monitoring System for Bridges			
Date	10 January 2013	Pages/Appendices	70+40
Supervisor(s) Mr. Seppo Karjalainen, Full Time Teacher			
Client Organisation/Partners Savonia University Of Applied Sciences			

Abstract

The aim of this thesis was to build a system that could measure force and vibration and send the measured data wirelessly. In addition, the purpose was to get acquainted with bridge condition monitoring systems. The thesis was done for Savonia University of Applied Sciences, School of Engineering and Technology.

The use of strain gauges and accelerometer was studied for the measuring system. The strain gauge measures the force and the accelerometer measures vibration. The Acam Picostrain PS081 microcontroller was used in strain gauge measurements. Texas Instrument MSP430 microcontroller was used in vibration measurements and wireless data transmission. The programming and use of the both microcontrollers was studied. After that necessary programs were made. A program, which collects measured data to the computer, was made for the data collection system.

The result of this thesis was a working system which measures vibration and sends measured data wirelessly to the computer.

Keywords
force measurement, vibration measurement

Public

ALKUSANAT

Tein opinnäytetyönäni järjestelmän, jolla pystyi mittaamaan voimaa ja värähtelyä. Lisäksi työhön kuului tutustuminen siltojen kunnonvalvontajärjestelmiin. Mittausjärjestelmä tehtiin siltojen kunnonvalvontaa varten. Työ tehtiin vuoden 2012 helmikuun ja syyskuun välisenä aikana. Työssä opin paljon sulautettujen järjestelmien ohjelmointia sekä erilaisten antureiden käyttöä. Tulevaisuutta ajatellen työ oli minulle tärkeä, koska elektroniikka-alalla ohjelmoinnin ja anturitekniikan osaaminen on hyvin tärkeää.

Kiitän opinnäytetyön ohjaajaa Seppo Karjalaista kaikesta avusta.

SISÄLTÖ

LIITTEET

LYHENTEET JA KÄSITTEET

AD-muunnin	(Analog-to-digital converter) Laite, joka muuntaa analogisen signaalin digitaaliseksi
Wheatstonen silta	Menetelmä vastuksen resistanssin määrittämiseksi
Piezosähköinen ilmiö	Kiteeseen syntyy sähköinen varaus, kun sen muotoa muutetaan
WLAN	Langaton lähiverkko
SPI	(Serial Peripheral Interface) Synkroninen sarjaliikenne väylä
RAM	(Random access memory) Laitteen käyttömuisti tai työmuisti
ROM	(Read Only Memory) Laitteen pysyväismuisti
EEPROM	(Electronically Erasable Programmable Read-Only Memory) Haihtumaton puolijohdemuisti
LCD	(Liquid Crystal Display) Nestekidenäyttö
I/O	(Input/Output) Sisään - tai ulostulo - pinni
LPM	(Low Power Mode) Vähävirtainen tila
FRAM	(Ferroelectric Random Access Memory) Kuten RAM, mutta käyttää ferrosähköistä-tekniikkaa
Mbps	(Megabits per seconds) Tiedonsiirtonopeus, joka ilmaisee kuinka monta megabittiä tieto on siirtynyt sekunnissa

Isäntätila	Laite, jonka käskyistä siihen liitetyt laitteet suorittavat halutut käskyt
Orjatila	Laite, joka suorittaa siihen liitetyn isäntälaitteen antamat käskyt
DMA	(Direct Memory Access) Tiedon siirtäminen muistiin ilman prosessorin käyttöä
Ajastin	(Timer) Kellolaite tai ohjelma, jonka avulla voidaan säädellä jonkun tapahtuman käynnistymis- ja päättymisaikoja.
PMW	(Pulse Width Modulation) Pulssinleveysmodulaatiolla pyritään säätämään kuormaan menevää jännitettä muuttamalla pulssisuhdetta. Tällä pyritään saamaan lähtösignaalin keskiarvo samaksi kuin modulointisignaalin arvo, joka on laskettu yhden värähtelyjakson aikana
LabView	National Instrumentsin tekemä graafinen ohjelmointiympäristö, joka perustuu G-kieleen
UDP	(User Datagram Protocol) Protokolla, joka ei tarvitse yhteyttä laitteiden välille, mutta mahdollistaa silti tiedonsiirron

1 JOHDANTO

Tässä opinnäytetyössä tutustutaan siltojen kunnonvalvontajärjestelmiin sekä antureihin, joita käytetään kunnonvalvontajärjestelmissä. Nämä anturit ovat venymäliuskat ja kiihtyvyysanturit. Lisäksi raportissa käydään läpi laiteohjelmointia kahdella eri laitteella. Opinnäytetyössä kehitetään mittauslaitteisto siltojen kunnonvalvontaa varten. Laitteistolla voidaan mitata voimaa venymäliuskoilla ja värähtelyä kiihtyvyysanturilla.. Näiden lisäksi mitattu data lähetetään langattomasti tietokoneelle. Laitteisto koostuu voiman- ja värähtelynmittausosasta, datansiirto-osasta ja datankeräysosasta. Voiman- ja värähtelyn mittaamiseen on olemassa erilaisia valmiita laitteistoja ja tässä työssä pyritään toteuttaa toimiva datankeräysjärjestelmä mittauslaitteiston rinnalle.

Siltojen kunnonvalvontajärjestelmä työ on jatkoa Sulautettujen järjestelmien erikoistyö -kurssilla tehtyyn työhön. Kurssi kuuluu elektroniikan koulutusohjelmaan ja työn aihe on saatu opettaja Seppo Karjalaiselta.

2 SILTOJEN KUNNONVALVONTA

Siltojen kunnonvalvonnalla tarkoitetaan siltojen rakenteellisen kunnon valvontaa ja tarkastusta erilaisilla sensoreilla (Gastineu, Johnson, & Schultz 2009, 13).
Siltojen kunnonvalvontaa tehdään, koska valmistumisen jälkeen silta altistuu monelle tekijälle, jotka vaurioittavat sillan kuntoa. Erilaisia vaurioittavia tekijöitä ovat ilmasto, hapettuminen, korroosio ja ikääntyminen. Lisäksi silta vaurioituu ajan myötä painotaakasta, joka syntyy liikenteestä. Koska edellä mainitut asiat lyhentävät sillan käyttöikää ja vaarantavat liikennettä, on siltaa varten rakennettava luotettava kunnonvalvontajärjestelmä. Sillan kunnonvalvontajärjestelmän avulla saadaan tietoa sillan kunnosta ajoissa ja voidaan ajoittaa sillan huoltoajankohdat.

Sillan kunnonvalvontajärjestelmä koostuu monesta eri osasta, joita ovat sensorijärjestelmä, datan kerääminen, datan siirtäminen, datan käsittely ja datan analysointi. Lisäksi järjestelmään kuuluu datan hallinnointi, etäkommunikointi ja datan arviointijärjestelmä. (Jing, Chun, Chun, & Liqiao 2009, 1 – 5.)

2.1 Sensorityypit

Sillan kunnonvalvonnassa mitataan erilaisilla sensoreilla eri suureita. Tässä työssä keskitytään eritoten värähtelyn, voiman ja taipumisen mittaamiseen tarkoitettuihin sensoreihin. Värähtelyn mittaamiseen käytetään kiihtyvyysanturia, koska värähtely voidaan kuvata kiihtyvyyden avulla. Voiman ja taipumisen mittaamiseen käytetään yleisesti erityyppisiä venymäliuska-antureita. Sillan siirtymän mittaamiseen on olemassa omat siirtymisen mittaamiseen tarkoitetut sensorit. (Gastineu ym. 2009, 15 – 16.)

2.2 Tehonsyöttö

Siltojen kunnonvalvontajärjestelmän sensoreille voidaan tuoda virtaa monella tavalla. Yleensä virta tuodaan suoraan sähköverkosta vaihtovirtana. Toinen vaihtoehto on käyttää akkuja. Aurinkoisilla paikoilla toimivat myös aurinkopaneelit, joilla voi ladata akkuja. (Gastineu ym. 2009.)

2.3 Datan keräys, siirto ja käsittely

Yleensä sillan kunnonvalvontajärjestelmissä käytetään langatonta sensoriverkkoa. Data kerätään sensoreilta langattomasti ja käsitellään analysointia varten. Käsittelyssä signaali AD-muunnetaan ja suodatetaan, jotta sitä olisi helpompi tutkia. Joissakin tilanteissa joudutaan signaalille käyttämään myös erilaisia muunnoksia. Yleensä kiihtyvyysanturin datalle tehdään DTFT-muunnos (Discrete-Time Fourier Transform), jolla data muutetaan aikatilasta taajuustilaan. Signaalinkäsittelyn jälkeen data voidaan tallentaa tietokantaan. (Zuozhou ym. 2012, 2 – 6.)

Datankeräysjärjestelmää suunniteltaessa on otettava huomioon, miten ja missä muodossa sensoreilta saatu signaali voidaan käsitellä ja analysoida. Sensorien signaalista täytyy saada selville, minkälainen signaali johtuu rasituksesta. Signaalissa voi olla erilaisia kohinoita, jotka vaikeuttavat signaalin analysointia. Tätä varten signaalia täytyy suodattaa. Tämän lisäksi datan siirrossa saattaa ilmetä ongelmia. Datapaketteja voi hävitä matkalle tai datan laatu on huonoa. (Jing ym. 2009, 2.)

2.4 Sillanvalvonnan esimerkkikohteita

2.4.1 Millau Viaduct

Yksi hyvä esimerkkikohde, jossa käytetään sillanvalvontaa, on Etelä-Ranskassa sijaitseva silta nimeltä Millau Viaduct. Tämä 2 460 metriä pitkä silta otettiin käyttöön vuonna 2004. Millau Viaduct on osa A75-moottoritietä, ja se rakennettiin ylittämään Tarn-joen laakso. Sillan eri osiin on asennettu lukuisia sensoreita mittaamaan sillan eri liikkeitä. Sensorijärjestelmä koostuu kiihtyvyysantureiden lisäksi tuulimittareista ja kallistumismittareista. Sillan venymää mitataan venymäsensoreilla. Sillan heilahtelut voidaan mitata kiihtyvyysantureilla millimetrien tarkkuudella. Data siirretään Ethernet-verkon avulla tietokoneille. (Millau Viaduct 2012.)

2.4.2 Rio-Antirio

Rio-Antirio on 2 880 metriä pitkä silta Kreikassa. Silta ylittää Korintinsalmen ja se otettiin käyttöön vuonna 2004. Tässä sillassa valvontajärjestelmä on toteutettu lukuisilla 3D-kiihtyvyysantureilla, venymäliuskoilla ja siirtymäantureilla. Rio-Antirio -sillan kunnonvalvonta on tärkeää, koska silta sijaitsee alueella, jossa maanjäristykset ovat mahdollisia. (Pararas-Carayannis 2007.)

KUVA 1. Rio-Antirion sillanvalvontajärjestelmää (Spectrum.Ieee 2012)

3 VOIMAN MITTAUS VENYMÄLIUSKOILLA

3.1 Venymäliuska

Venymäliuska on laite, jolla voidaan mitata voimaa. Lisäksi sillä voidaan mitata myös rasitusta rakenteissa ja materiaaleissa. (Yongdae ym. 2010, 1.)
Venymäliuskat reagoivat mekaaniseen rasitukseen ja niissä on aina jokin elektroninen parametri, joka muuttuu rasituksen voimasta (Stefanescu 2011, 1).

Venymäliuskoja on olemassa neljää eri tyyppiä. Yleisin venymäliuskatyyppi on resistanssin muutokseen perustuva venymäliuska, koska se on tarkin ja edullisin vaihtoehto. Muita venymäliuskatyyppejä ovat akustinen, optinen ja mekaaninen venymäliuska. Resistanssisen venymäliuskan toimintaperiaate perustuu resistanssin muutokseen venymäliuskassa, kun siihen kohdistuu voima. Resistanssin muutoksen avulla voidaan laskea materiaalin kohdistuvan voiman suuruus. (Yongdae ym. 2010, 1.)

Venymäliuskan resistanssi voidaan laskea kaavalla:

$$R = \rho \cdot \frac{L}{A} \tag{1}$$

Tässä R on venymäliuskan resistanssi ja ρ on resistiivisyys. L on venymäliuskan pituus ja A materiaalin pinta-ala. (Lidan ym. 2007, 1.)

Rasituksen tekijä GF on mekaanisesta rasituksesta johtuvan elektronisen resistanssin muutoksen suhde. GF ilmaisee myös venymäliuskan tarkkuuden. Se voidaan laskea kaavalla:

$$GF = \frac{(R(\varepsilon) - R_0)/R_0}{\varepsilon} \tag{2}$$

Tässä R(ε) on venymäliuskan resistanssi venymisrasituksen ollessa ε. R_0 on venymäliuskan resistanssi, kun rasitus on 0 ja ε on venymäliuskaan kohdistuva mekaaninen venymisrasitus. (Yongdae ym. 2010, 4.)

Venymisrasitus ε saadaan laskettua materiaalin alkuperäisen pituuden ja venymisestä johtuvan pituuden suhteesta:

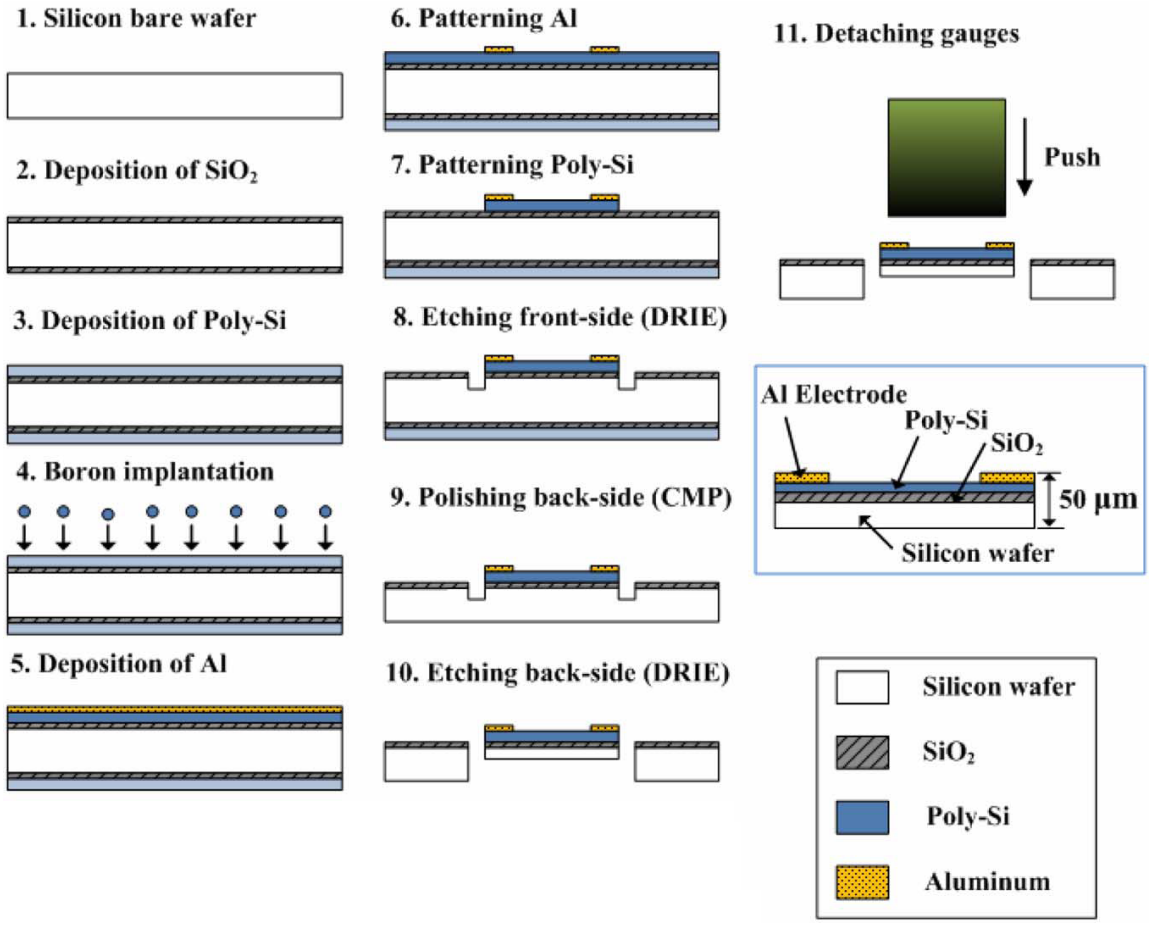
$$\varepsilon = \frac{\Delta L}{L} \tag{3}$$

Resistanssin muutokseen perustuvia venymäliuskoja on olemassa kolmea eri tyyppiä: lanka-, metallikalvo- ja polypiivenymäliuskat. Yleisimmin näistä on käytössä metallikalvovenymäliuska, mutta nykyaikaisin ja tarkin venymäliuskatyyppi on polypiivenymäliuska. Polypiivenymäliuskoja on olemassa joustavia ja ei-joustavia. Ei-joustavaa venymäliuskaa ei pysty suoraan liimaamaan mitattaviin rakenteisiin. Tämän takia kehitettiin joustavat polypiivenymäliuskat, jotka voidaan liimata suoraan mitattavaan alustaan. Polypiivenymäliuskoja käytetään erilaisissa anturiratkaisuissa, kuten paine-, kiihtyvyysantureissa sekä gyroskoopeissa. (Yongdae ym. 2010, 1.)

3.2 Venymäliuskan valmistus ja rakenne

Kalvovenymäliuskan valmistuksessa metallikalvo syövytetään ruudukolle, joka on sähköisesti eristetyllä hartsilla (Kyowa 2012, 6).

Polypii-venymäliuskassa on yleensä voileipämäinen rakenne. Rakenne koostuu kolmesta kerroksesta, joita ovat eristävä kerros, polypii-kerros ja ei-johtava metallinen kerros. Polypii-venymäliuskan valmistusprosessi on kuvattu vaihe vaiheelta kuvassa 2. Valmistus aloitetaan tyhjästä piikiekosta, joka päällystetään ensiksi pii-dioksidilla ja boron-ioneilla. Sitten kiekko kerrostetaan alumiinikerroksella. Tämän jälkeen alumiini- ja pii-oksidi-kerrokset kuvioidaan kiekon päälle. Lopuksi kerrokset syövytetään oikeaan muotoon ensiksi kiekon etupuolelta ja sitten takapuolelta. Syövytyksessä käytetään etsausta. (Yongdae ym. 2010, 2 – 3.)



KUVA 2. Polypii-venymäliuskan valmistus (Yongdae ym. 2010, 3)

3.3 Venymäliuskan liimaus

Venymäliuskojen liimauksessa käytetään yleensä lasijuottamista. Tällä tavalla venymäliuskoihin saadaan matala hystereesi. (Yongdae ym. 2010, 2.)

Venymäliuskojen kiinnitystapa riippuu venymäliuskan tyypistä. Yksinkertaisin liimaus tehdään kalvovenymäliuskalle. Ensiksi alusta puhdistetaan huolellisesti jollakin kemiallisella aineella. Sitten venymäliuska asetetaan alustaan liiman kanssa. Venymäliuskan paikalleen painamisen ajaksi liuskan päälle on laitettava teippiä. Kun liima on kuivunut, teippi poistetaan ja voidaan kiinnittää venymäliuskan johdot. (Wilson 2005, 523.)

3.4 Wheatstonen silta

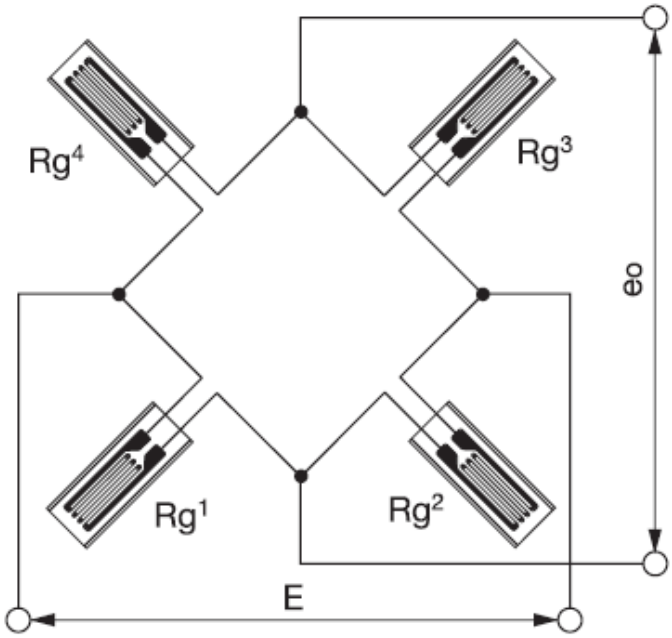
Venymäliuskojen rasituksesta johtuva elektronisen parametrin muutos on yleensä hyvin pieni ja vaikeasti havaittava. Tämän takia venymäliuskat rakennetaan Wheatstonen siltaan. Wheatstonen silta muuntaa muuttuvan resistanssin muuttuvaksi jännitteeksi. (Kyowa 2012, 9.)

Wheatstonen silta koostuu yleensä neljästä venymäliuskasta, jotka kaikki mittaavat rasitusta. Siltaan ajetaan sisään käyttöjännitettä ja ulostuleva jännite on mittaussignaali. Käyttöjännitteen ja ulostulevan signaalin suhde voidaan laskea kaavalla 4.

$$\frac{U_E}{U_A} = \frac{R_1}{R_{1+}R_2} - \frac{R_4}{R_{3+}R_4} \quad (4)$$

Tässä U_E on ulostuleva signaali ja U_A on käyttöjännite. R_1, $R_{1,}$ R_1 ja R_4 ovat Wheatstonen sillan venymäliuskat. (Stefanescu 2011, 1.)

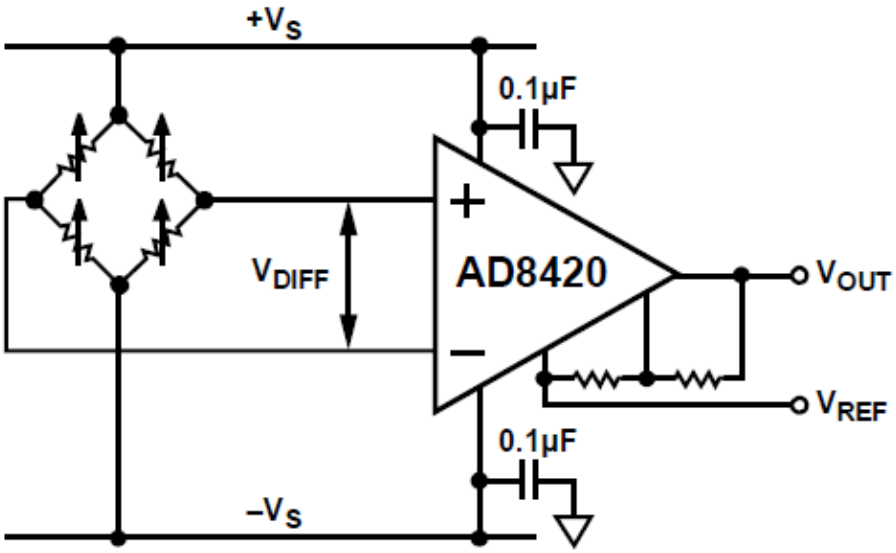
Wheatstonen sillasta voidaan käyttää myös puolikasta siltaa ja neljännesosasiltaa. Puolikkaassa sillassa on kaksi venymäliuskaa ja neljännesosasillassa on vain yksi venymäliuska. Kuvassa 3 on kokonainen Wheatstonen silta, jossa on neljä mittaavaa venymäliuskaa kytketty siltaan. Kuvassa kirjain E tarkoittaa sillan käyttöjännitettä ja e_0 on sillan ulostulojännite.



KUVA 3. Wheatstonen silta (Kyowa 2012, 7)

3.5 Signaalin vahvistaminen

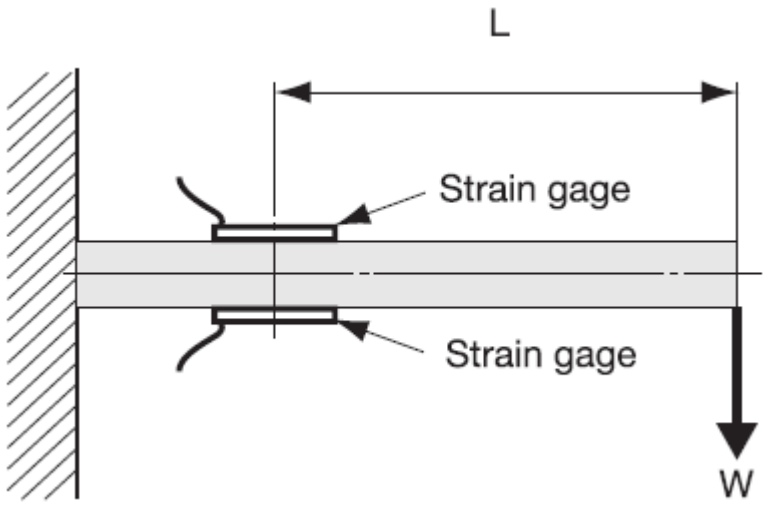
Siltakytkennästä saatu signaali pitää vahvistaa ennen kuin se voidaan viedä jatkokäsiteltäväksi. Wheatstonen silta kytketään esimerkiksi instrumentointivahvistimeen, joka vahvistaa signaalin riittävän suureksi. Kuvassa 4 on Analog Devicen instrumentointivahvistin AD8420, joka on liitetty siltakytkentään.



KUVA 4. Instrumentointivahvistin ja siltakytkentä (Analog Devices 2012, 25)

3.6 Mittaus venymäliuskoilla

Venymäliuskalla voidaan mitata monella eri tavalla rasitusta ja voimaa. Mittaustapa riippuu käytössä olevasta Wheatstonen sillan tyypistä. Esimerkiksi, jos mitataan tiettyyn palkkiin aiheutuvaa rasitusta, niin neljännesosasillassa venymäliuska kiinnitetään vain palkin yhdelle puolelle. Puolikkaassa sillassa venymäliuskat kiinnitetään molemmin puolin palkkia. Kuvassa 5 on esimerkkimittaus puolikkaalla Wheatstonen sillalla. Siinä kaksi venymäliuskaa on kiinnitetty palkin ylä- ja alapuolelle. Kuvassa kirjain L tarkoittaa etäisyyttä venymäliuskan keskikohdasta palkin päähän. Kirjain W tarkoittaa palkkiin kohdistuvaa voimaa. (Kyowa 2012, 8.)



KUVA 5. Mittaus puolikkaalla Wheatstonen sillalla (Kyowa 2012, 8)

Puolikkaan sillan ratkaisussa pinnan rasitus σ voidaan laskea kaavalla 5.

$$\sigma = \frac{\varepsilon}{2} \times \mathrm{E} \qquad (5)$$

Tässä ε on rasitus ja E sillan käyttöjännite. (Kyowa 2012, 8.)

4 VÄRÄHTELYN MITTAUS KIIHTYVYYSANTURILLA

4.1 Kiihtyvyysanturi yleisesti

Kiihtyvyysanturi on anturi, jolla mitataan kiihtyvyyttä, värähtelyä ja iskuja. Kiihtyvyysanturia käytetään yleisesti monessa asiassa, kuten tutkimuksissa ja erilaisissa mittauksissa. Sitä käytetään myös eritoten autoteollisuudessa. Yleisimmin käytössä oleva kiihtyvyysanturi on piezosähköinen kiihtyvyysanturi. Muita kiihtyvyysanturityyppejä ovat piezoresistiivinen- ja kapasitiivinen kiihtyvyysanturi. (Wilson 2005, 137.)

4.2 Piezosähköisen kiihtyvyysanturin toimintaperiaate

Piezosähköinen kiihtyvyysanturi sisältää piezosähköinen osan, joka toimii sensorin mittaavana elementtinä. Piezosähköinen osa on yleensä jonkinlainen vieteri anturin sisällä. Lisäksi sensori sisältää seismisen massan. Piezosähköinen vieteri yhdistää anturin pohjan ja seismisen massan toisiinsa. Kun anturin pohjaan kohdistuu kiihtyvyys, piezosähköisessä materiaalissa syntyy voima. Tämän voiman suuruus riippuu siitä, kuinka suuri kiihtyvyys on tai kuinka suuri seisminen massa on.

Voima voidaan laskea Newtonin I:n lain mukaan:

$$F = ma \tag{6}$$

Tässä m on anturin sisällä oleva seisminen massa ja a anturiin kohdistuva kiihtyvyys. (Wilson 2005, 138.)

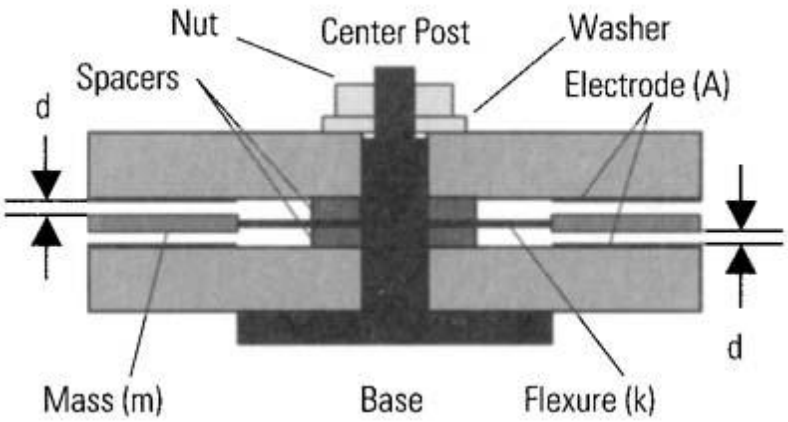
4.3 Piezoresistiivisen kiihtyvyysanturin toimintaperiaate

Piezoresistiivisessä kiihtyvyysanturissa on piezoresistiivisiä vastuksia, joiden vastus muuttuu, kun kiihtyvyysanturiin kohdistuu kiihtyvyys. Piezoresistiivinen kiihtyvyysanturi sisältää palkin, jonka toisessa päässä on tietyn suuruinen massa. Massa taivuttaa palkkia kiihtyvyyden mukaan ja palkissa olevat vastukset reagoi-

vat tähän taipumiseen. Tämän lisäksi vastukset on laitettu Wheatstonen siltaan, jotta ulostulevasignaali olisi jännitteen muutos. (PCB Piezotronics 2012.)

4.4 Kapasitiivisen kiihtyvyysanturin toimintaperiaate

Kapasitiivisen kiihtyvyysanturin toimintaperiaate on samantapainen kuin piezoresistiivisen kiihtyvyysanturin. Kapasitiivisessa kiihtyvyysanturissa muuttuva suure on kapasitanssi. Kiihtyvyysanturin sisällä on vastakkaiset levykondensaattorit, joiden etäisyydet vaihtelevat kiihtyvyyden mukaan. Kuvassa 6 on esillä kapasitiivisen kiihtyvyysanturin toimintaperiaate. (Wilson 2005, 146.)



KUVA 6. Kapasitiivisen kiihtyvyysanturin toimintaperiaate (Wilson 2005, 146)

4.5 Kiihtyvyysanturin valmistus

Nykyisin valmistus- ja pakkaustapana käytetään MEMS (Micro-Electro Mechanical Systems) -tekniikkaa. MEMS-tekniikan avulla voidaan yhdistellä mekaanisia, elektronisia ja optisia asioita. MEMS-tekniikassa on paljon etuja verrattuna vanhoihin tekniikoihin. Laitteista saadaan pienikokoisia, kevyitä, vähävirtaisia ja edullisia. (Rogers, Ramadoss, Ozmun & Dean 2007, 1.)

MEMS-anturit valmistetaan etsaamalla eli syövyttämällä kerroksia piialustan päälle (Gardner, Varadan & Awadelkarim 2001, 399).

4.6 Kiihtyvyysanturin signaalin käsittely

Kiihtyvyysanturin signaali on analogista ja se pitää muuntaa digitaaliseksi, jotta signaali voidaan käsitellä. Signaalin muuntaminen tehdään AD-muuntimella, joka voi olla erillinen piiri tai osana mikroprosessoria. Kiihtyvyysanturin signaali täytyy myös suodattaa, jotta päästäisiin eroon syntyneestä kohinasta, joka epäselkeyttää heikkoa signaalia. Kohina voidaan suodattaa esimerkiksi alipäästösuodattimella. Alipäästösuodatin voidaan toteuttaa ohjelmistolla mikroprosessorissa tai yksinkertaisella RC-piirillä, jossa on vastuksen ja kondensaattorin kytkentä. Digitaalisen suodattimen yhtälö voi olla esimerkiksi kaavan 7 mukainen.

$$\mathrm{H(z)} = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} B_k z^{-k}}{1 - \sum_{k=1}^{N} A_k z^{-k}} \tag{7}$$

Tästä saadaan muodostettua sisään ja ulostulon korrelaatio:

$$\mathrm{y(n)} = \sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k) \tag{8}$$

(Ming, Xiaokun & Yawen 2008, 3.)

4.7 Värähtelyn mittaus

Kiihtyvyysanturi soveltuu värinän mittaamiseen sen hyvän taajuusvasteen takia. Yksinkertaisimmillaan värinänmittausjärjestelmään kuuluu kiihtyvyysanturin lisäksi signaalin käsittelyosa, signaalin analysointiosa ja laitteisto, jolla signaali tallennetaan. Signaalin käsittelyosassa signaalia vahvistetaan ja muutetaan impedansseja. Analysointiosassa signaali muunnetaan haluttuun muotoon ja lopuksi signaalin tallennusosassa signaali tallennetaan. (Morris & Langari 2012, 523 – 524.)

5 LANGATON DATANSIIRTO

Tässä työssä käytettiin langattomaan datansiirtoon WLAN-tekniikkaa. WLAN eli wireless LAN tarkoittaa langatonta lähiverkkoa, jonka avulla voidaan liittää laitteita toisiinsa langattomasti. Ajan myötä WLAN-tekniikka on kehittynyt niin, että nopeudet ovat nousseet ja muutenkin tekniikka on parantunut. WLAN:lle on määritelty oma IEEE (Institute of Electrical and Electronic Engineers) standardinsa.

Ensimmäinen IEEE-standardi oli 802.11. Sen maksimi tiedonsiirtonopeus on 2 Mb/s. Seuraava standardi oli 802.11b, ja sen tiedonsiirtonopeus voi olla maksimissaan 11 Mb/s. Nämä kaksi standardia toimivat molemmat 2.4 Ghz:n taajuudella.
Uudemmat standardit ovat 802.11a ja 802.11g. Näissä tiedonsiirtonopeus voi olla 54 Mb/s ja taajuus 5 GHz. 802.11n on uusin käytössä oleva standardi. Se yhdistää 2.4:n ja 5:n GHz:n käyttötaajuudet. 802.11n käyttää MIMO (Multiple-inputs, Multiple-outputs) -tekniikkaa. MIMO-tekniikka pystyy käyttämään useampaa antennia ja pystyy näin ollen käsittelemään enemmän dataa kerralla.

Langattomassa tiedonsiirrossa voi olla heikkouksia verrattuna langalliseen tiedonsiirtoon. Yksi heikkous on niin sanotut kuolleet pisteet. WLAN-verkossa voi olla niin sanottuja kuolleita kohtia, joissa signaali on heikko tai sitä ei ole ollenkaan. Nämä kuolleet kohdat verkossa voivat johtua muista samantaajuisista signaalilähteistä. (Caro 2008, 1 – 6, 9 – 10.)

Langaton verkko täytyy suojata salasanalla, jottei sitä pääse muut käyttämään luvattomasti. Salaustyyppejä ovat WEP (Wireless Equivalent Protocol) ja WPA (Wi-Fi Protected Access). WEP-salauksessa käytetään 40/64-bittisiä salasanoja, jotka eivät kuitenkaan ole kovin varmoja. WPA-suojaus on tehty korvaamaan WEP-suojaus ja parantamaan salasanojen varmuutta. Lisäksi WPA-suojaus lisäsi tunnistettavien käyttäjien kapasiteettia. (Chen 2005, 999.)

6 ENERGIAHARVESTERIT

Energiaharvestereilla tarkoitetaan laitteita, joilla kerätään pieniä määriä energiaa ympäristöstä. Energianlähteinä voivat olla esimerkiksi tuulivoima, aurinkovoima, lämpö tai tärinä. Harvesterin tuottamalla energialla on pyritty korvaamaan kannettavien laitteiden paristot ja muut virtalähteet. Energian keräämiseen on olemassa erilaisia sensoreita ja toimintatapoja. Yksi esimerkki on piezosähköinen energiaharvesteri, joka muuntaa mekaanisen energian sähköiseksi energiaksi. (Mhetre, Nagdeo & Abhyankar 2011, 1.)

Energiaharvestereita käytetään seuraavissa teollisuuden sovelluksissa:

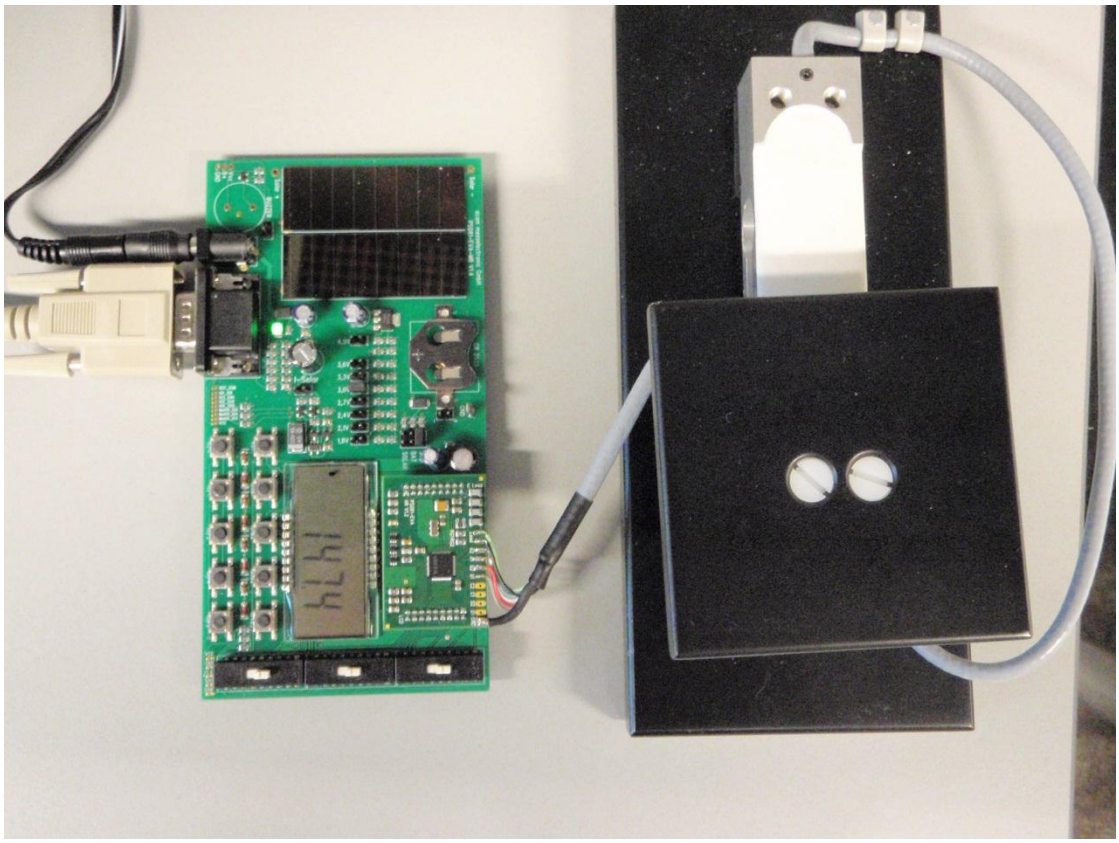
- potilaiden etäseuranta
- valvonta ja turvallisuus
- maatalouden sovellukset
- kotiautomaatio
- rakenteiden seuranta. (Texas Instruments 2012e.)

7 DB_PS081-EVA-KIT

Tässä työssä perehdyttiin Acamin valmistamaan venymäliuskaprosessoriin PS081. Venymäliuskaprosessorin avulla oli tarkoitus tehdä työhön liittyvät voiman ja rasituksen mittaukset.

PS081-EVA-KIT koostuu emolevystä, kuormitusanturista, kolmesta prosessorimoduulista ja PICOPROG-ohjelmointilaitteesta. Prosessorimoduulit sisältävät PS081-prosessorin ja ne liitetään emolevy-alustalle. Moduuleita käytetään eri käyttötarkoitusta varten. Ensimmäinen moduuli on korkearesoluutio-moduuli, jota käytetään kun halutaan saada PS081-prosessorista täysi teho irti. Toinen moduuli on nimeltään standard-moduuli. Sitä käytetään vähemmän korkearesoluutioisissa-sovelluksissa. Kolmas moduuli on nimeltään Wheatstone-moduuli ja sitä käytetään, kun halutaan kytkeä Wheatstonen silta venymäliuskakorttiin kiinni. (Acam 2012a, 5.)

Kitin mukana tuleva kuormituskenno kytketään korkearesoluutio-moduuliin. Kuvassa 7 on PS081-EVA-KIT ja mukana tuleva kuormituskenno toimintavalmiina.



KUVA 7. PS081-EVA-KIT

Normaalissa käytössä käytetään standard-moduulia. Siihen voidaan kytkeä sensoreita kuten muihinkin moduuleihin. Lisäksi moduuleissa on juotettu valmiiksi lämpötilasensorit lämpötilamittausta varten. Emolevykortilla on LCD-näyttö, josta voidaan lukea esimerkiksi mittaustuloksia. Lisäksi kortille voidaan liittää ulkoinen LCD-näyttö.

Laitteessa on muun muassa seuraavia ominaisuuksia:

- 10 painonappia
- aurinkokenno
- SPI-liitäntä
- paristo
- RS232-liitin. (Acam 2012a, 5.)

7.1 Acam Picostrain PS081 -siltavahvistinprosessori

Acam Picostrain PS081 on 24-bittinen mikroprosessori, joka on suunniteltu käytettäväksi erilaisissa painonmittaussovelluksissa. Lisäksi PS081:lla voidaan mitata voimaa ja vääntömomenttia. (Acam 2012b, 6.)

7.1.1 RAM, ROM ja EEPROM

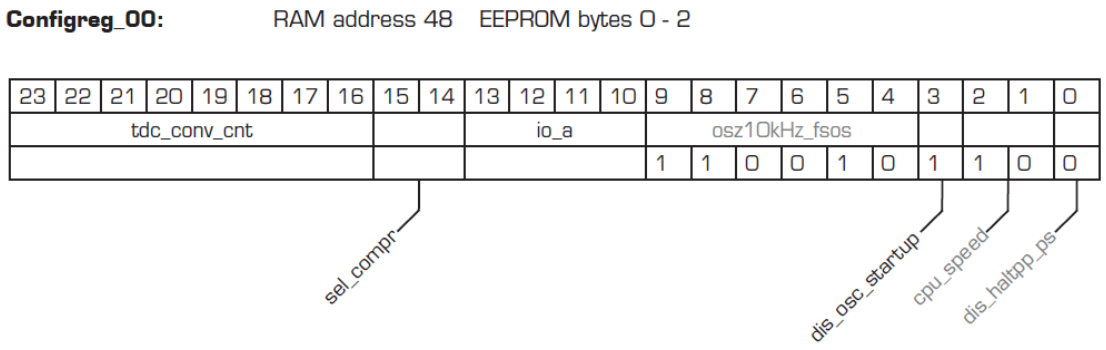
Picostrain PS081 -prosessorikortti sisältää RAM-, ROM- ja EEPROM-muistia. RAM-muistin osoiteväylä sisältää 127 osoitetta. Jokaisen RAM-muistin osoitteen osoitin on kartoitettu ALU-rekisteriin. RAM-muistin osoitteen osoitinta voidaan hallita ramad-komennolla.

127 osoitteesta muistipaikat 0 – 15 on varattu käyttäjän omiin tarpeisiin. Muistipaikat 16 – 31 sisältävät esimerkiksi venymäliuskojen mittausdataa. Muistipaikat 32 – 47 ovat myös käyttäjälle tarkoitetut. Konfiguraatiorekisterit sijaitsevat muistipaikoissa 48 – 66. Tästä eteenpäin on varjomuistia ja LCD-näytön segmenttien tarvitsemaa muistia.

ROM-muisti on pysyvää muistia, jossa sijaitsee oleellisimmat käskyt ja asiat. Näitä asioita tarvitaan esimerkiksi mittauksissa. ROM-muisti alkaa muistipaikasta 2 048. EEPROM-muistin koko on 2 048 tavua, ja se on jaettu neljään 512 tavun lohkoon. (Acam 2012b, 104 – 106.)

7.1.2 Konfiguraatiorekisterit (Configuration Registers)

PS081-mikroprosessori sisältää 18 konfiguraatiorekisteriä, jotka ovat 24 bitin levyisiä. Konfiguraatiorekisterit sijaitsevat muistipaikasta 48 muistipaikkaan 66. Konfiguraatiorekistereillä ohjataan koko laitetta ja kaikkia siihen liitettyjä asioita, kuten LCD -ohjainta ja venymäliuskojen mittausta. PS081-prosessorikortin asetukset on peilattuna EEPROM-muistissa. Kun käytetään kortin omaa prosessoria, asetukset kirjoitetaan EEPROM-muistiin. Ulkoista prosessoria käytettäessä asetukset kirjoitetaan RAM-muistiin. Kuvassa 8 on esillä konfiguraatiorekisteri 11, jossa sijaitsee esimerkiksi kortin I/O-porttien hallinta. Kuvan taulukon ylärivillä on ilmoitettu bitin numero ja alarivillä bitille asetettu arvo. Konfiguraatiorekisterien käytöstä kerrotaan lisää ohjelmointiosiossa luvussa 7. (Acam 2012b, 88.)



KUVA 8. Esimerkki konfiguraatiorekisteristä (Acam 2012b, 97)

7.1.3 Venymäliuskaratkaisut

Picostrainin muuntimessa on portit erilaisia venymäliuskaratkaisuja varten. Näitä ovat esimerkiksi:

- neljän itsenäisen puolikkaan sillat
- kaksi puolikasta siltaa, jotka muodostavat kokonaisen sillan
- kaksi itsenäistä puolikasta siltaa
- 1 klassinen Wheatstonen silta
- yksi yksittäisen puolikkaan silta.

Venymäliuskojen rasituksenmittaus perustuu purkautuvan ajan mittaukseen. Purkautuva aika määritellään venymäliuskan resistanssilla ja kondensaattorin kapasitanssin

kuormituksella. Kuormituksen arvo saadaan määritettyä kahden purkautuvan ajan suhteesta. Tarkkuus on luokkaa 15 ps:n resoluutiolla. (Acam 2012b, 24.)

7.2 Muita ominaisuuksia

PS081-venymäliuskaprosessorikortissa on myös muita hyödyllisiä ominaisuuksia, joita voidaan käyttää venymäliuskamittausten yhteydessä.

7.2.1 Kuormakondensaattori

Kuormakondensaattorilla on iso vaikutus mittauksen laatuun ja lämpötilan vakauteen. Picostrain:n valmistajan mukaan Cload eli kuormakondensaattori pitää valita venymäliuskojen resistanssin mukaan. Cload voidaan laskea kaavalla:

$$Cload = 0{,}7 \times Rsg \times Cload \quad (9)$$

jossa Rsg on venymäliuskojen resistanssi. (Acam 2012b, 32.)

7.2.2 Kellojakso

Kellojaksolla tarkoitetaan aikaväliä seuraavien purkautuvien ajanmittausten välissä. Purkautuva aika määritellään venymäliuskojen ja kuormakapasitanssin arvojen perusteella. Suositeltu purkautuva aika on väliltä 80 – 150 us. Lisäksi latausajan on oltava tarpeeksi pitkä takaamaan kuormakondensaattorin uudelleen latauksen täydeksi asti. Kellojakso voidaan asettaa Picostrainin rekisteriin.

Purkautumisjaksot määräytyvät sen mukaan, minkälainen venymäliuskaratkaisu on käytössä. Esimerkiksi puolikkaalla sillalla tarvitaan kaksi jaksoa ja kokonaisella sillalla neljä jaksoa. (Acam 2012b, 33.)

7.2.3 Tilat ja ajoitukset

Venymäliuskaprosessori PS081 sisältää kolme erilaista toimintatilaa. Näitä ovat jatkuva tila, yksittäisen muunnoksen tila ja venytetty tila. Jatkuvassatilassa prosessori tekee jatkuvaa purkautuvan ajan mittausta. Yksittäisen muunnoksen tilassa prosessori tekee täydellisen mittaussarjan ja menee sitten lepotilaan. Venytetty tila yhdistää

muutaman mittauksen edut. On myös olemassa jatkuva venytetty tila, joka yhdistää jatkuvan tilan ja venytetyn tilan. Tässä tapauksessa puolikkaiden siltojen purkautuvan ajan mittausten välillä on pitempi väli. Jatkuvan venytetyn tilan lisäksi voidaan käyttää jatkuvaa yksittäistä muunnosta.

Eri tilojen käyttö riippuu siitä, mihin venymäliuskoja käytetään. Muunnostila valitaan prosessorin konfiguraatiorekisterissä kaksi. Jos halutaan yksittäinen muunnos, asetetaan käsky single_conversation = 1. Jatkuvassa tilassa valitaan 0. Venytetty tila asetetaan konfiguraatiorekisteriin kolme käskyllä stretch. (Acam 2012b, 37 – 39.)

7.2.4 Jälkikäsittely

Kun mittaus on tehty, muunnin tekee jälkikäsittelyn mitatulle datalle. Muunnin tallentaa kalibroidun ja skaalatun tuloksen laitteen RAM-muistin rekisteriin. (Acam 2012b, 47.)

7.2.5 Oskillaattori

Laite sisältää sisäisen vähävirtaisen 10 kHz:n oskillaattorin. Sitä käytetään erilaisissa ajastin-funktioissa ja kellojakson määrittämisessä.

Oskillaattorin toiminta määritellään konfiguraatiorekisteriin numero kolme. PS081 sisältää myös oskillaattoriohjauksen ulkoiselle 4 Mhz:n resonaattorille. Sitä voidaan käyttää esimerkiksi ajan mittauksessa. (Acam 2012b, 58.)

7.2.6 LCD-näyttö

PS081-prosessori sisältää LCD:n ohjauksen. LCD-ohjaukselle on varattu 18 pinniä, joilla voidaan lähettää maksimissaan kuusi numeroa, pilkun ja kahdeksan erikoismerkkiä. Korttiin voidaan liittää myös ulkoinen LCD-ohjain SPI-väylän avulla. (Acam 2012b, 59.)

7.3 Liitännät

PS081-prosessorissa on I/O-pinnit erilaisia liitäntöjä varten. Sarjamuotoisen datan lähetys tehdään SPI-liitännällä, joka toimii I/O-pinnien avulla.

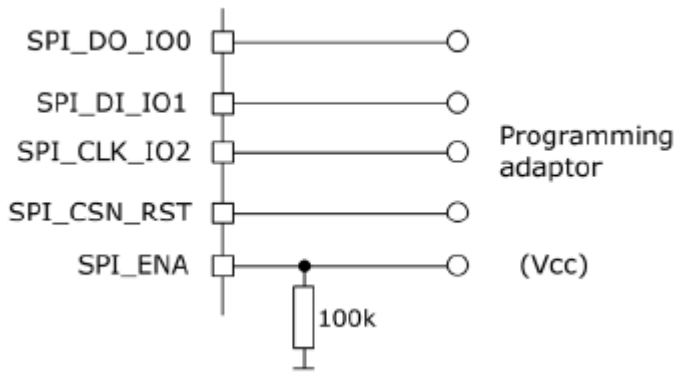
PS081 sisältää 6 I/O–pinniä. I/O–pinnit on nimetty seuraavasti:

SPI_DO_IO0
SPI_DI_IO1
SPI_CLK_IO2
MULT _IO3
MULT _IO4
MULT _IO5

SPI_DO on ulostulo sarjamuotoiselle datalle ja SPI_DI on sisääntulo. SPI_CLK on sarjamuotoinen kellosignaali. Mult-alkuiset pinnit ovat monikäyttöisiä I/O-pinnejä. I/O-pinnit määritetään konfiguraatiorekistereissä 11 ja 17.

Rekisteriin määritetään luku 11, jos halutaan pinnistä sisääntulon. Ulostulo määritetään luvulla 00. (Acam 2012b, 72 – 73.)

SPI-väylän avulla PS081-prosessori voidaan liittää toiseen mikrokontrolleriin. SPI-väylän avulla voidaan lähettää erilaista tietoa eteenpäin tai vaikkapa hallita toista laitetta. PS081:n SPI-liitännät ovat kuvan 9 mukaiset.



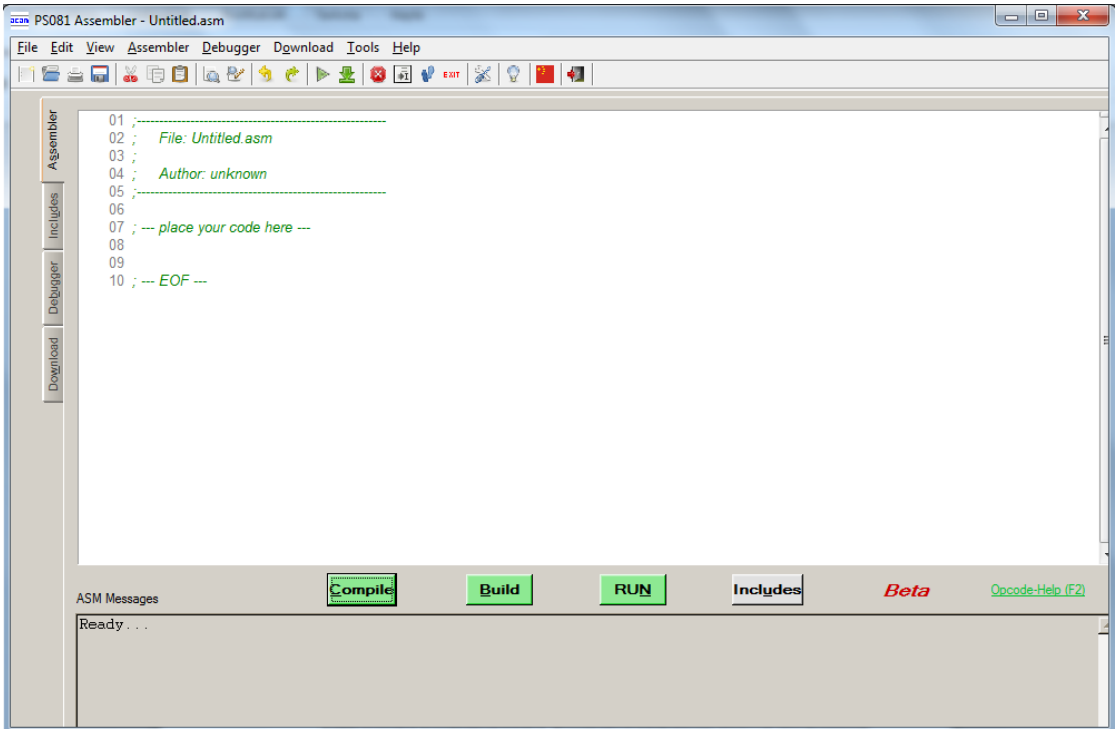
KUVA 9. SPI-väylän liitännät (Acam 2012b, 75)

SPI-väylä saadaan käyttöön asettamalla SPI_ENA-pinni ylätilaan. Kun SPI-väylä on aktiivinen, liitännät toimivat SPI-portteina normaalien I/O-porttien sijaan.

SPI-väylää hallitaan omilla komennoilla, jotka ovat hexamuotoisia assembler-käskyjä. Kaikki käskyt kirjoitetaan RAM-muistiin ja sieltä ne myös luetaan. SPI-käskyt voidaan kirjoittaa myös EEPROM-muistiin, mutta valmistaja suosittelee käyttämään RAM-muistia, kun käytetään ulkoista mikroprosessoria. (Acam 2012b, 75 – 81.)

8 PICOSTRAIN PS081 -OHJELMOINTI

PS081-prosessorin ohjelmointi tehdään laitteen omassa Assembler-ohjelmointiympäristössä (kuva 10). Tässä työssä oli tarkoituksena tutustua laitteen mittauksen ohjelmointiin sekä SPI-väylän ohjelmointiin. PS081-mikrokontrolleripaketti sisältää valmiin mittausohjelmiston sekä useita esimerkkiohjelmia.



KUVA 10. PicoStrain-ohjelmointiympäristö

PS081-ohjelmointi on assembler-kielellä tehtävää ohjelmointia ja se sisältää lähes kaikki tunnetut konekieliset käskyt. Konekieliset käskyt tosin riippuvat aina kyseessä olevasta prosessorista. Käskyt voivat olla eri nimillä. Ohjelmointiympäristö sisältää peruskäskyt kuten move, add, sub jne.

PS081-prosessorissa on kolme erilaista akkua, joihin voidaan tallentaa tietoa. Nämä ovat x-akku, johon tallennetaan yleensä mittausdata. Sitten on y-akku, johon voidaan

tallentaa esimerkiksi lämpötilan arvot. Kolmas akku on z-akku, joka toimii lippurekisterinä.

Vakiot ja muuttujat määritellään komennolla CONST. Näiden määrittely tehdään aina ohjelman alussa. Vakio määritellään esimerkiksi seuraavalla tavalla:

CONST result_HB0 20

Vakion perässä oleva luku viittaa vakion muistipaikan numeroon.

Ehkä tärkein komento tässä ohjelmointiympäristössä on ramadr. Tällä komennolla asetetaan osoitin tietylle muistiosoitteelle. Seuraavassa esimerkissä osoitetaan mittaustuloksen muistipaikkaan result_HB0 ja siirretään sen arvo x-akkuun.

Ramadr result_HB0
move x, r

(Acam 2012b, 106 – 110.)

8.1 Konfiguraatiorekisterin ohjelmointi

Konfiguraatiorekisteri on ohjelmoinnin kannalta tärkeimpiä asioita PS081-prosessorissa. Rekistereitä on 18 kappaletta ja niillä ohjataan laitteen kaikkia toimintoja. Joka rekisterissä on 24 bittiä, joita voi muuttaa 1:ksi tai 0:ksi. Kun bitin arvo on yksi, se on käytössä ja nolla, kun se on pois käytöstä. Jokainen bitti konfiguraatiorekisterissä vastaa jotain toimintoa. Kuten esimerkiksi I/O-porttia tai LCD-näytön jotain asetusta. Jokaisen konfiguraatiorekisterin kaikki 24-bittiä lasketaan yhteen hexadesimaalisena. Saadut arvot ilmoitetaan ohjelman config.h-tiedostossa, joka sisällytetään pääohjelmaan.

Konfiguraatiorekisterit ilmoitetaan config.h-tiedostossa seuraavalla tavalla:

equal 0x2C828A ; Config Register 0

Tässä luku 0x2C828A on yhteenlaskettu hexadesimaalinen luku konfiguraatiorekisteri 0:sta.

Konfiguraatiorekisterin arvoja voidaan muuttaa kesken pääohjelman tai pelkästään config.h-tiedostossa. Pääohjelmassa muuttaminen onnistuu esimerkiksi bitin muuttamisella nollasta ykköseksi ja toisinpäin. Tai sitten muuttaa suoraan koko hexadesimaalisen luvun seuraavalla tavalla:

```
ramadr      config_reg_0
move r,     0x378280 ;Config Register 0
```

Tässä konfiguraatiorekisteri 0:aan muutetaan koko luku.
Bitin muuttaminen tapahtuu komennolla bitset. Jos haluaa bitin kokonaan nollaksi, käytetään komentoa bitclr. (Acam 2012b, 88 – 101.)

8.2 SPI-väylän ohjelmointi

SPI-väylän ohjelmointi tapahtuu hexadesimaalisilla komennoilla. Kun halutaan PS081-mikroprosessori SPI-tilaan, SPI_ENA-pinni pitää asettaa ylätilaan. SPI-ohjelma aloitetaan resetoimalla kortti käskyllä 0xF0. Tämän käskyn nimi on power reset. Sitten asetetaan vahtikoira pois päältä komennolla 0x09.

Sitten konfiguroidaan RAM-muisti. Se tapahtuu komennolla 0x00, joka tarkoittaa RAM Write:ä, eli muistiin kirjoitusta.

Tässä kohdassa täytyy kirjoittaa RAM-muistiin osoitevälille 48 – 66. Konfiguroinnin jälkeen alustetaan kortti komennolla 0xC0 (Init Reset) ja aloitetaan uusi sykli komennolla 0xCC (Start new cycle).

Tämän jälkeen SPI_D0-pinni pitää niin sanotusti ”pollata” eli tehdään jatkuva silmukka, jossa kortti tarkistaa onko pinni valmiina. Uusi lähettävä data saadaan lähetettyä, kun SPI_D0 menee tilasta yksi tilaan nolla. Kun SPI_D0 menee tilasta yksi tilaan nolla, tällöin pinni SPI_CS täytyy asettaa myös tilasta yksi tilaan nolla. Näin SPI-yhteys saadaan toimimaan. Sitten mittaustulos luetaan RAM-muistin osoitteesta nolla komennolla 0x40 (Ram Read). (Acam 2012b, 75 – 81.)

8.3 Mittausohjelma

Acam Picostrain PS081 -venymäliuskakortin mukana tulee kattava esimerkkiohjelmisto venymäliuskamittauksiin ja muihin kortilla tehtäviin mittauksiin. Jos käyttää laitteen omaa venymäliuska-anturia, siihen löytyy valmiita esimerkkiohjelmia. Tässä työssä käytettävän SPI-väylän esimerkkiohjelmaa ei tule kortin mukana, vaan se pitää itse ohjelmoida.

8.4 Mittausohjelman rakenne

Toimiva mittausohjelma koostuu monesta osasta. Aluksi asetetaan CONST-muuttujat. Sitten aloitetaan itse ohjelma, jossa ensimmäiseksi asetetaan laite käynnistymään resetistä funktiossa rst_pwr_check. Tämän jälkeen mennään program_entry-funktioon, jossa määritetään, mihin funktioon mennään seuraavaksi. Tässä tarkistetaan, mikä bitti on käytössä. Jos bitti on yksi, tila on alustustilassa. Jos bitti on kaksi, mennään mittausfunktioon. Bitti nollalla mennään sleep-tilaan.

Seuraavaksi hypätään valittuihin funktioihin, joita on siis init-funktio, jossa asetetaan kaikki data ja mittausfunktio, jossa suoritetaan mittaus. Get_initial_offset-funktiossa otetaan mittaukselle alkuarvo. Ohjelmaan voidaan sisällyttää myös automaattinen käynnistyminen auto_on-funktiolla. Display-funktiossa näytetään mitattu data laitteen LCD-näytöllä. Get_initial_offset-funktiota varten täytyy laskea mitatun datan mediaani erillisissä median-funktioissa.

Ohjelman tärkein funktio on measurement-funktio, jossa suoritetaan varsinainen mittaus. Mitattu arvo tallentuu result_HB0-muuttujaan, joka sijaitsee rekisterissä 20. Measurement-funktiossa suoritetaan tarvittavat laskut, jotta mitatusta luvusta saataisiin järkevä. Kun luku on muutettu oikeaan muotoon, se tallennetaan muuttujaan last_display_value. Tämä muuttuja voidaan suorittaa display-funktiossa ja näyttää laitteen LCD-näytöllä. Konfiguraatiorekisteriin täytyy asettaa myös oikeat asetukset, jotta ohjelma toimisi oikein. Nämä asetukset menevät config-tiedostoon. Liitteessä 1 on esillä voimanmittauksen esimerkkiohjelma.

9 MSP430-MIKROKONTROLLERI

MSP430-mikrokontrolleria käytettiin tässä työssä värähtelyn mittauksessa ja langattomassa tiedonsiirrossa. Texas Instrumentsin MSP430 -mikrokontrollerit ovat vähävirtaisia 16-bittisiä mikroprosessoreita. Prosessorit perustuvat helppokäyttöiseen RISC-arkkitehtuuriin. MSP430-mikrokontrolleri sisältää muun muassa seuraavia asioita:

- 10/12-bittinen AD-muunnin
- komparaattori
- DMA (Direct Memory Access)
- ajastimia
- liitännät: SPI, UART, I^2C
- I/O - pinnien määrä välillä 14 – 113.

Lisäksi jotkin MSP430-mallit sisältävät myös muita ominaisuuksia.

MSP430-mikrokontrollerin kellotaajuus on 8 – 25 MHz riippuen mallista. Muisti on Flash- tai RAM-muistia. Flash-muistin koko on 0.5 – 256 kB ja RAM-muistia voi olla 128 B – 18 kB. Tässä työssä käytetty MSP430-malli FR5739 sisältää FRAM-muistia. FRAM-muisti on huomattavasti nopeampaa kuin Flash-muisti. FRAM-muistin tiedonsiirtonopeus voi olla 2 MB/s, kun taas Flash-muistin siirtonopeus voi olla maksimissaan 12 kB/s. FRAM-muistin koko on 8 – 16 kB riippuen mallista. MSP430-mikrokontrollerit ovat hyvin vähävirtaisia laitteita. Ne sisältävät erilaisia virransäästötiloja (LPM), jotka voivat keskeytyksien avulla sammuttaa prosessorin, kun sitä ei tarvita. (Texas Instruments 2012a, 2 – 12.)

Tässä työssä käytettiin Texas Instrumentsin valmistamaa MSP430FR5739-mikrokontrolleria (kuva 11). Se sisältää 10-bittisen AD-muuntimen sekä edellä mainitun FRAM-muistin. FRAM-muistin koko on 16 kt. Kontrollerissa on liitäntä WLAN-moduulille, jota tarvitaan tässä työssä. Tässä MSP430-kortissa on sisäänrakennettuna myös kiihtyvyysanturi. (Texas Instruments 2012b.)

MSP430FR5739-mikrokontrollerin maksimi kellotaajuus on 24 MHz. Kuitenkin FRAM-muistia käytettäessä kellotaajuus on pudotettava 8 MHz:iin tai muuten tiedonsiirtoon tulee viiveitä. Tässä työssä on tärkeää, että laitteet ovat vähävirtaisia ja sen takia on hyvä tietää jotain laitteiden virrankulutuksista. Valmistajan ilmoittama virrankulutus on 81.4 µA / MHz. Kun käytetään 24 MHz:a ja RAM-muistia, virrankulutus on tyypillisesti 1,45 – 1,75 mA. FRAM-muistin kanssa kulutus on n. 2,2 mA. LPM0-virransäästötilassa virrankulutus on 175 µA, kun lämpötila on 25 °C. Kun käytetään LPM2-, LPM3- tai LPM4-tiloja, virtaa kuluu 5,9 – 80 µA. Kaikkein vähävirtaisimmat

tilat ovat LPM3.5 ja LPM4.5. Näissä virtaa kuluu 1,5 – 2,2 µA ja 0,32 – 66 µA. (Texas Instruments 2012c, 1 & 46 – 48.)

KUVA 11. MSP430FR5739 (Texas Instruments 2012b)

10 WLAN-MODUULI CC3000 TIWI-SL

Työssä käytettiin LS Researchin valmistamaa WLAN-moduulia CC3000 TiWi-SL. Moduuli on IEEE 802.11 b/g -yhteensopiva ja maksimitiedonsiirtonopeus on 54 Mbps. Moduulissa on SPI-liitäntä, jolla se liitetään mikrokontrolleriin. WLAN-moduuli on kiinnitetty omaan alustaansa, jossa on antenni ja kiinnityspinnit MSP430FR5739-mikrokontrolleriin. Moduuli ja alusta ovat esillä kuvassa 12. (LS Research 2012, 1 & 20.)

KUVA 12. TiWi-LS-WLAN –moduuli (Texas Instruments Processors Wiki 2012a)

11 MSP430FR5739-OHJELMOINTI

Tässä osiossa käydään läpi MSP430-sarjan mikrokontrollerin MSP430FR5739:n ohjelmointia.

MSP430-prosessorikortin ohjelmointi voidaan tehdä Code Composer Studio -ohjelmalla tai IAR Kickstart -ohjelmalla. Ohjelmointikielinä voi käyttää C:tä tai Assemblyä. Ohjelmointikielenä käytettiin C:tä, koska se on tutumpi ja helppokäyttöisempi.

Tässä työssä oli tarkoituksena opetella ohjelmoimaan MSP430-kortin SPI-väylää ja AD-muunninta. Lisäksi piti opetella DMA:n ja ajastimen käyttöä. MSP430-mikrokontrolleri sisältää myös muita ohjelmoitavia ominaisuuksia, mutta niitä ei tässä työssä tarvittu.

11.1 I/O-porttien ohjelmointi

MSP430-prosessorikortin I/O-pinnejä ohjataan I/O-rekistereillä. Ohjaus tapahtuu asettamalla rekisterin biteille haluttu arvo. Esimerkiksi pinnien sisään- ja ulostuloja ohjataan PxIN- ja PxOUT -rekistereillä. Jos halutaan bitti käyttöön eli tila ylös, niin asetetaan bitille arvo yksi. Bitille asetetaan arvo nolla, kun halutaan bitti pois käytöstä eli bitin tila alas. Muita I/O-porttien ohjausrekistereitä on:

PxDIR = bitin suunnan valinta. 0 = sisääntulo, 1 = ulostulo
PxREN = ylös- tai alasvetovastuksen valinta
PxSEL0 ja PxSEL1 = funktion valintarekisterit
PxIFG = portin keskeytysrekisteri
PxIE = keskeytyksen valinta
PxIES = keskeytyksen reunan valintarekisteri
PxIV = keskeytysvektori.

MSP430-prosessorikortin pinnit voidaan ilmaista kahdella tavalla, kun niitä ohjelmoidaan. Esimerkiksi pinni 1 voi olla joko BIT1 tai hexadesimaalisena 0x001.
Esimerkki pinnin suunnan valinnasta:
P1DIR |= BIT1;
Tässä merkki |= tarkoittaa OR-operandia, joka asettaa bitin ykköseksi. (Texas Instruments 2012d, 275 – 277.)

11.2 SPI-väylän ohjelmointi

MSP430-mikrokontrollerissa on yleensä kaksi SPI-väylää. Nämä ovat nimeltään eUSCI_A ja eUSCI_B. MSP430-sarjan mikrokontrollerien käskyt ja rekisterit eroavat vähän toisistaan. Tässä työssä perehdytään MSP430FR5739-mikrokontrollerin SPI-väylän ohjelmointiin. Mikrokontrolleri voidaan asettaa SPI-väylän isännäksi tai orjaksi.

SPI-väylän ohjelmoinnissa käytetään SPI-väylän ohjausrekistereitä UCA0CTLW0 ja UCB0CTLW0. SPI-väylä käyttää MSP430-kortin normaaleja pinnejä, mutta pinnit on kuitenkin nimetty erilailla kuin normaalisti. SPI-pinnit ovat UCxSIMO, UCxSOMI, UCxCLK ja UCxSTE.

Isäntätilassa UCxSIMO-bitti on ulostulolinja ja orjatilassa sisääntulolinja. UCxSOMI-bitti taas on isäntätilassa sisääntulo ja orjatilassa ulostulo. UCxCLK-bitti eli kellosignaali on isäntätilassa ulostulo ja orjatilassa sisääntulo. UCxSTE-bittiä käytetään neljän johdon kytkennässä, jos tarvitaan monta isäntä- tai orjakorttia. UCxSTE-bitillä kytketään orjakortin datansiirto päälle.

Datan siirrossa käytetään datasiirto-puskuria UCxTXBUF. Dataa siirrettäessä datasiirtopuskurissa oleva tieto siirtyy datarekisteriin TX. Puskurista siirretyn datan ilmaisee siirtokeskeytyslippu UCTXIFG. Datan vastaanottopuskuri on nimeltään UCxRXBUF. Siinä on myös samanlainen keskeytyslippu vastaanotetulle datalle nimeltään UCRXIFG. (Texas Instruments 2012d, 458 – 460.)

11.2.1 Neljän ja kolmen pinnin isäntätila

SPI-tilaksi voidaan valita neljän tai kolmen pinnin SPI-tila. Neljän pinnin tiloja on olemassa kaksi erilaista. Pinnin tila asetetaan rekisterillä UCSTEM. Jos rekisterin arvoksi asetetaan yksi, niin pinnit UCxSIMO ja UCxCLK ovat sisäänmenoja. Tässä tilassa datan siirto riippuu UCxSTE-pinnistä.

Toisessa neljän pinnin tilassa UCSTEM rekisterin arvo asetetaan nollaksi. Tässä tilassa orjakortin datansiirron toimintaan kytkevä signaali menee automaattisesti toimintatilaan. Kolmen pinnin isäntätila on muuten sama, kuin neljän pinnin tila, mutta siinä ei käytetä orjan valitsemiseen käytettävää UCxSTEM-pinniä. (Texas Instruments 2012d, 461 – 462.)

11.2.2 Neljän pinnin orjatila

Orjakortti saa isäntäkortilta signaalin UCxSTEM-pinnistä ja silloin orjakortin datansiirto kytkeytyy päälle. USCI-moduuli kytketään päälle nollaamalla UCSWRST-bitti. Kun tämä on tehty, USCI on valmis datansiirtoon ja vastaanottoon. Isäntätilassa kellosignaali on valmis ja orjatilassa se on pois päältä. Kellosignaali valitaan UCSSELx-bitillä ja se tuotetaan eUSCI-bitin kellogeneraattorissa. (Texas Instruments 2012d, 462.)

11.2.3 SPI-väylän alustus

SPI-väylän alustus aloitetaan määrittämällä pinnit. Kolmen pinnin tilassa käytetään sisään- ja ulostuloa sekä kellosignaalia. Neljän pinnin tilassa käytetään näiden lisäksi orjan aktivointipinniä. Yleisimmin MSP430-mikrokontrollerin SPI-väylän ulostulona käytetään pinniä 2.0 ja sisääntulona 2.1. Kellosignaali-pinninä voidaan käyttää esimerkiksi pinniä 1.5.

Pinnien määrityksen jälkeen aloitetaan komentojen asettaminen SPI-väylän rekistereihin. Ensimmäisenä on hyvä asettaa SPI-väylä toimimaan mikrokontrollerin reset –painikkeesta. Tämä tapahtuu komennolla UCSWRST. Sitten asetetaan SPI-väylälle haluttu tila, joko kolmen pinnin tai neljän pinnin SPI-tila. Samalla voidaan asettaa SPI-väylä synkroniseksi komennolla UCSYNC. Ja jos halutaan, että SPI-väylä lähettää ensimmäisenä eniten merkitsevän bitin, asetetaan komento UCMSB. SPI-väylän kellon toiminta asetetaan komennolla UCSSELx.
Muita SPI-väylän komentoja ovat:

- UCA0BR0 = bitin suhteen valinta
- UCA0MCTLW = modulaation valinta.

SPI-väylässä kannattaa käyttää keskeytysperiaatetta. MSP430-mikrokontrolleriin tämä asetetaan keskeytysrekisteriin UCAxIE tai UCBxIE. Keskeytysrekisteriin asetetaan joko lähetyskeskeytys tai vastaanottokeskeytys. Lähetyskeskeytys on nimeltään UCTXIE ja vastaanotto UCRXIE. (Texas Instruments 2012d, 462 – 470.)

11.3 SPI-väylän lähetys ja vastaanotto

Kun alustukset on saatu tehtyä, siirrytään datan lähettämiseen ja vastaanottoon. Lähetys ja vastaanotto voidaan toteuttaa tavallisessa funktiossa tai keskeytysrekisterin avulla toteutetussa funktiossa. Molemmissa tavoissa käytetään keskeytyspalvelulippua UCAxIFG, mutta eri tavoin. Keskeytyslippurekisterin lisäksi tarvitaan lähetys- tai vastaanottokeskeytysliput. Normaalissa funktiossa tehdään while-silmukka, joka toteutuu, jos lähetys- tai vastaanottopuskuri on valmis. Tämä voidaan toteuttaa seuraavalla tavalla:

while (!(UCA0IFG&UCTXIFG));

Tässä silmukka toteutuu, jos keskeytyslippurekisteri UCA0IFG ja lähetyksen keskeytyslippu UCTXIFG ovat nollia. Kun silmukka toteutuu, voidaan mennä eteenpäin. Sitten data saadaan siirrettyä SPI-väylään, kun määritetään, että lähetyspuskuri on sama kuin lähetettävä data. Tämä tehdään seuraavalla tavalla:

UCA0TXBUF = TXData;

Tässä UCA0TXBUF on lähetyspuskuri ja TXData on haluttu lähetettävä data.
Vastaanotto tapahtuu samalla tavalla, mutta siinä käytetään vastaanoton keskeytyslippua while-silmukassa. Lisäksi lähetyspuskurin sijaan käytetään UCA0RXBUF-vastaanottopuskuria.

Toinen vaihtoehto lähettämiselle ja vastaanotolle on keskeytysvektorirekisterin käyttäminen. Tässä vaihtoehdossa prosessori menee niin sanottuun virransäästötilaan erilliseen funktioon. Aina kun tarvitaan uusi lähetys tai vastaanotto, prosessori kytkeytyy toimintatilaan. Ohjelma menee tähän funktioon komennolla
__bis_SR_register(LPM0_bits+GIE) ja tulee pois komennolla
__bic_SR_register_on_exit(CPUOFF).

Tätä funktiota kutsutaan ISR-funktioksi. Siellä ohjelma valitsee niin sanotun kytkimen avulla, mikä suoritus tehdään seuraavaksi. SPI-väylässä toimintoja on kaksi, lähetys ja vastaanotto. (Texas Instruments 2012d.)

11.4 AD-muuntimen ohjelmointi

MSP430-mikrokontrolleri sisältää 10- tai 12-bittisen AD-muuntimen. AD-muuntimen moduulin nimi on ADC10_B tai ADC12_B. MSP430-kortin AD-muuntimen ohjausrekisterit ovat nimeltään ADC10CTL0, ADC10CTL1 ja ADC10CTL2. Jos AD-muunnin on 12-bittinen, rekisterin nimessä on luku 12 luvun 10 sijaan. AD-muunnin kytketään toimimaan komennolla ADC10ON. Muunnos saadaan käyntiin komennolla ADC10ENC, ja muunnettu data tallentuu ADC10MEM0-rekisteriin.

Muunnoksen kaava muunnetulle datalle N_{ADC} saadaan laskettua kaavalla 10.

$$N_{ADC} = 1023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}} \quad (10)$$

Tässä V_{R+} ja V_{R-} ovat valittavissa olevat jänniterajat, joilla asetetaan muunnoksen ylä- ja alarajat (Texas Instruments 2012d, 396 – 398.)

11.4.1 AD-muuntimen alustus

AD-muuntimen ohjelmointi aloitetaan asettamalla halutut asetukset AD-muuntimen rekistereihin. Ensimmäisenä varmistetaan, että komento ADC10ENC on tyhjennetty. Sitten määritetään AD-muunnin toimintakuntoon komennolla ADC10ON ja asetetaan haluttu syklien lukumäärä muunnoksessa komennolla ADC10SHTx. Kirjain x komennon perässä merkitsee tiettyä numeroa, joka halutaan asettaa. Numero on binäärinen luku, joka merkitsee eri vaihtoehtoa ja toimintoa tietylle komennolle. Nämä edellä mainitut komennot asetetaan ohjausrekisteriin ADC10CTL0.

ADC10CTL1-rekisteriin asetetaan seuraavat komennot:

- ADC10SHSx: muunnoksen lähteen valinta
- ADC10SHP: muunnoksen pulssin valinta
- ADC10CONSEQx: muunnoksen järjestyksen valinta
- ADC10SSELx: kellon valinta.

ADC10CTL2-rekisteriin asetettavat komennot:

- ADC10RES: muunnoksen resoluution valinta.

Ohjausrekistereihin voidaan asettaa myös muita komentoja, mutta edellä mainitut ovat yleisimmät ja tärkeimmät.

Ohjausrekistereiden lisäksi MSP430-mikroprosessorissa on myös muunnoksen muistirekisteri. Sinne asetetaan muunnoksen referenssijännite komennolla ADC10SREFx ja muunnoksen sisääntulokanavat komennolla ADC10INCHx. Jos halutaan käyttää keskeytyspalvelua, asetetaan komento ADC10IE.
(Texas Instruments 2012d, 398 – 401.)

11.4.2 AD-muuntamisen ohjelmointi

Kun alustukset on tehty, voidaan aloittaa AD-muuntaminen. Tämä voidaan tehdä normaalisti ikisilmukassa tai keskeytysperiaatteen mukaan. Normaalilla tavalla tehtäessä muunnos aloitetaan komennolla ADC10ENC. Lisäksi samalla asetetaan komento ADC10SC, joka aloittaa varsinaisen muunnoksen. Näiden jälkeen tehdään while-silmukka, jossa on AD-muuntimen keskeytysrekisteri ADC10IFG ja komento ADC10IFG0, joka indikoi, onko muunnos valmis. Kun nämä molemmat ovat nollia, muunnos on valmis ja silmukan ehto toteutuu. Seuraavana esimerkki tästä silmukasta:

```
while ((ADC10IFG & ADC10IFG0)==0);
```

Kun ehto on toteutunut, AD-muunnettu data on AD-muuntimen muistirekisterissä ADC10MEM0. While-silmukkaan voidaan valita myös toiset rekisterit, joilla indikoidaan muunnoksen tilaa. ADC10BUSY-komento kertoo muunnoksen tilan.

Toinen tapa tehdä muunnos on käyttää keskeytysperiaatetta. AD-muuntimen keskeytysbitti ADC10ISR kytkeytyy automaattisesti, kun muunnos on ladattu ADC10MEM0-rekisteriin. MSP430-mikrokontrolleri siirtyy näin ollen vähävirtaiseen tilaan ja kytkeytyy pois siitä komennolla __bis_SR_register(CPUOFF + GIE). Prosessori kytkeytyy taas käyntiin, kun suoritetaan uusi muunnos. Niin sanotun kytkimen avulla määritetään, mikä toiminto suoritetaan. Tässä tapauksessa kytkin valitsee eri keskeytyslipun AD-muunnokselle. (Texas Instruments 2012d.)

11.5 DMA-ohjelmointi

DMA:n avulla voidaan siirtää tietoa osoitteesta toiseen ilman prosessoria. Esimerkiksi AD-muunnoksen tulokset voidaan siirtää suoraan RAM-muistiin. DMA-ohjain sisältää kahdeksan eri kanavaa, joita voidaan käyttää tiedon siirtämiseen paikasta toiseen. DMA-ohjain sisältää neljä erilaista käsittelytilaa, joilla määritetään, miten data siirretään. Näitä ovat:

- kiinteästä osoitteesta kiinteään osoitteeseen
- kiinteästä osoitteesta lohkon osoitteeseen
- lohkon osoitteesta kiinteään osoitteeseen
- lohkon osoitteesta lohkon osoitteeseen.

Käsittelytiloja ohjataan DMASRCINCR- ja DMADSTINCR-ohjausbiteillä. DMASRCINCR-bitillä valitaan, onko lähdeosoite suurennettu, pienennetty vai ennallaan. DMADSTINCR-bitillä tehdään samat asiat, mutta kohdeosoitteelle.

DMA:ssa on kuusi erilaista siirtotilaa. Näitä ovat:

- yksittäinen siirto
- lohkosiirto
- purske-lohkosiirto
- jatkuva yksittäinen siirto
- jatkuva lohkosiirto
- jatkuvan purskeen -lohkosiirto.

Siirtotila valitaan DMADT-rekisterillä. DMA:n liipaisin valitaan DMAxTSEL-komennolla. Liipaisimena voi toimia esimerkiksi AD-muunnin, joka kytkee DMA:n toimimaan, kun muunnos on valmis.

DMA kytketään toimintakuntoon DMAEN-komennolla ja sammutetaan myös samalla komennolla. Rekisterin tila vaihdetaan tilasta yksi tilaan nolla. DMA:lle on myös olemassa keskeytyspalvelu ja se kytketään DMAIE-rekisterillä. (Texas Instruments 2012d, 250 – 269.)

11.6 Ajastimen ohjelmointi

Ajastinta tarvitaan, jotta ohjelmassa olevia asioita voidaan suorittaa ja lopettaa oikeaan aikaan.

MSP430 sisältää kaksi eri ajastinta Timer_A ja Timer_B. Molemmat moduulit sisältävät samat ominaisuudet ja rekisterit. Ajastinta voidaan käyttää tallennuksessa, vertailussa, laskurina ja ulostulon PWM:n ohjauksessa. Ajastinta voidaan käyttää myös keskeytysperiaatteella.

Timer-moduulit sisältävät neljä rekisteriä, joihin komennot asetetaan. Rekisterit ovat:

- ohjausrekisteri TAxCTL
- laskentarekisteri TAxR
- kaappaus-vertailurekisteri TAxCCTLn
- keskeytysvektorirekisteri TAxIV.

Ohjausrekisteriin asetetaan muun muassa ajastimen kellon lähde komennolla TASSEL. Muita komentoja ovat MC, TACLR ja TAIE. MC-komennolla asetetaan ajastimen tila ja TACLR-komennolla nollataan ajastin. TAIE-komennolla asetetaan keskeytys käyttöön ja pois käytöstä.

Ajastimen aika asetetaan komennolla TA0CCR0. MC-komennolla asetetaan, onko laskuri ylöspäin tai alaspäin laskeva. Laskuri voi olla myös jatkuvassa tilassa.
Kaappaus-vertailurekisteriin voidaan asettaa esimerkiksi kaappaustila ja ulostulon tila. Tällä rekisterillä voidaan siis ohjata ulostuloa komennolla OUTMOD.
(Texas Instruments 2012d, 310 – 327.)

12 MSP430-MIKROKONTROLLERIN AD-MUUNTAMINEN

Tässä työssä yksi tärkeimmistä asioista on AD-muuntaminen. Sensoreilta saatu signaali on analogista ja se täytyy muuntaa digitaaliseksi, jotta sitä voidaan jatkokäsitellä. Tämän takia työssä keskityttiin paljon AD-muuntamiseen. Lisäksi AD-muuntamisesta oli saatava riittävän nopeaa. Tavoitteena oli saada vähintään 10 kilonäytettä sekunnissa tapahtuvaa muunnosta. Muunnoksen oli vielä tapahduttava kolmelta eri kanavalta. Vaikka tavoitteena oli käyttää virransäästö-ominaisuuksia, niitä ei työssä käytetty.

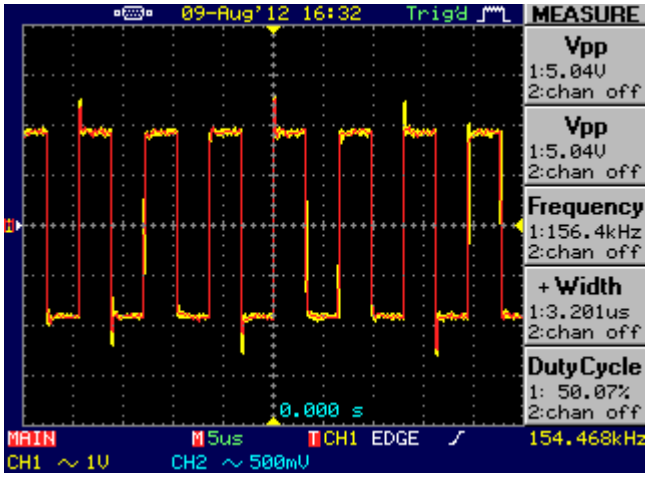
12.1 Muunnos yhdeltä kanavalta

Kun AD-muuntimen toiminta oli opiskeltu, alettiin tehdä AD-muuntamista MSP430-mikrokontrollerilla. AD-muunnosta voidaan tehdä useammalta kanavalta samanaikaisesti, mutta aluksi muunnosta tehtiin vain yhdeltä kanavalta. Testiohjelmasta tehtiin hyvin yksinkertainen ohjelma, joka tekee jatkuvaa AD-muuntamista ikisilmukassa. Ohjelman aluksi asetetaan AD-muuntimelle oikeat asetukset ja sen jälkeen on while-silmukka, jossa tapahtuu AD-muunnos. Tässä tapauksessa käytettiin erillistä TakeAdcMeas-funktiota, joka pyörii ikisilmukassa (kuvio 13).

```
void TakeAdcMeas(void)
{
  ADC10CTL0 |= ADC10ENC | ADC10SC;
  while ((ADC10IFG & ADC10IFG0)==0);    // onko muunnos valmis?
  P1OUT ^= BIT0;
}
```

KUVIO 13. AD-muunnoksen funktio

Ensiksi muunnos käynnistetään ADC10ENC- ja ADC10SC-komennoilla ja sitten ADC10IFG0-rekisterin avulla todetaan, onko muunnos valmis. MSP-kortin sisääntulopinni yksi vaihtaa tilaa aina, kun muunnos on valmis. Tämän avulla voidaan seurata muunnosnopeutta oskilloskoopilla. Saatu muunnosnopeus on esillä kuvassa 14. Liitteessä 2 on esillä esimerkkiohjelma AD-muuntamisesta.



KUVA 14. AD-muuntimen muunnosnopeus

Asetuksista muunnosnopeudeksi saatiin 150 - 200 kilonäytettä sekunnissa.

12.2 AD-muunnos useammalta kanavalta

Tarkoituksena oli saada AD-muunnettua kolmelta eri kanavalta, mitä varten ohjelmaan piti tehdä joitain muutoksia. I/O-pinnit täytyi määrittää kolmelle sisääntulolle ja AD-muuntimen sisääntulokanavien määrä piti nostaa. Useamman kanavan muunnos on esillä tämän työn luvussa 14.

12.3 AD-muunnoksen lähetys SPI-väylään

Työssä kokeiltiin myös muunnoksen lähettämistä SPI-väylään. Ohjelmaan piti määrittää SPI-väylän pinnit SIMO, SOMI ja CLK. Tässä tapauksessa käytössä oli vain ulostulopinni SIMO. Tämän jälkeen ohjelmaan lisätään SPI-väylän vaatimat asetukset ja lähetysfunktio, joka lähettää muunnoksen SPI-väylään.

Yksikertaisimmillaan SPI-väylän lähetysfunktio näyttää tältä:

```
TXData = ADCResult;
while (!(UCA0IFG&UCTXIFG));        // Onko USCI_A0 TX puskuri valmis?
UCA0TXBUF = TXData;                // Lähetetään data
```

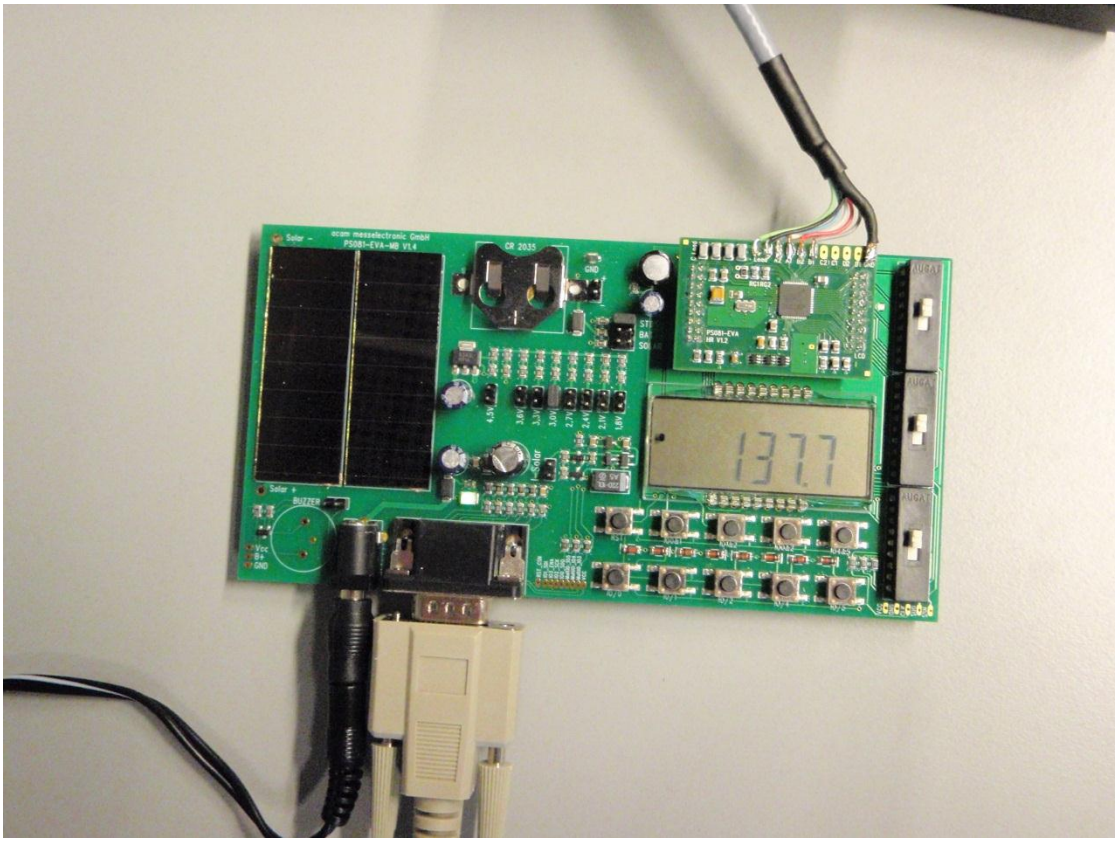
Ensiksi määritellään, että lähetettävä data on AD-muunnoksen tulos. Sitten tarkastetaan, onko lähetyspuskuri valmis ja sen jälkeen data lähetetään puskuriin.
Liitteessä 3 on esimerkkiohjelma AD-muunnoksen lähettämisestä SPI-väylään.

13 LAITTEEN KUVAUS

Työn tarkoituksena oli tehdä mittauslaitteisto siltojen kunnonvalvontaa varten. Laitteisto mittaa voimaa, rasitusta ja värähtelyä sekä lähettää datan eteenpäin langattomasti. Laitteen tuli olla pienikokoinen ja vähävirtainen. Data tuli saada lähetettyä reaaliaikaisesti tietokoneelle datankeräysohjelmaan.

14 VENYMÄLIUSKAOHJELMAN TESTAUS

Picostrain PS081 -venymäliuskaprosessorikortin mukana tulee kattava ohjelmisto venymäliuskamittauksia varten. Tämän raportin liitteessä 1 on esimerkki mittausohjelmasta, joka käyttää venymäliuskakitin kuormituskennoa voiman mittaamiseen. Ohjelma mittaa kuormituskennoon aiheutuvat painallukset ja näyttää tuloksen PS081-kortin LCD-näytöllä. Kuvassa 15 on esillä mittaustilanne, jossa mitattu tulos näkyy LCD-näytöllä.



KUVA 15. Venymäliuskaohjelman testaus

15 AD-MUUNNOKSEN LÄHETYS LANGATTOMASTI MSP430-MIKROKONTROLLERILLA

Tarkoituksena oli saada mitattua värähtelyä kiihtyvyysanturilla ja lähettää saatu data langattomasti tietokoneelle. Ensiksi keskityttiin AD-muunnoksen lähetykseen, jotta langaton lähetys saataisiin toimimaan. Tavoitteena oli saada tehtyä vähintään 10 kHz:n näytteenottoa kiihtyvyysantureilta ja lähettää se eteenpäin. MSP430-mikrokontrolleri kykenee muuntamaan yhdeltä kanavalta yli 200 kilonäytettä sekunnissa. Oikeilla AD-muuntimen asetuksilla maksiminopeus voidaan saavuttaa. 3-akselinen kiihtyvyysanturi sisältää kolme kanavaa, joten muuntamisen on tapahduttava kolmelta eri kanavalta. Näin ollen muunnosnopeus on jaettava kolmella, jotta saadaan todellinen nopeus.

15.1 MSP430FR5739-mikroprosessorin ohjelma

Työssä käytettiin MSP430FR5739-mikroprosessorille tehtyä WLAN-moduulia. Texas Instrumentsin sivuilta löytyy kattava ohjelmistopaketti tälle moduulille. Tämän työn kannalta sivuilta löytyi datalogger-ohjelma, joka kerää dataa kiihtyvyysanturilta. Ohjelma lähettää kiihtyvyysanturin dataa tietokoneella olevalle ohjelmalle ja tallentaa datan tekstitiedostoon. Datalogger-ohjelma koostuu WLAN-lähetysosasta ja AD-muunnoksen osasta. MSP430-kortille tuleva valmis ohjelma olisi muuten sopinut suoraan tähän tarkoitukseen, mutta siinä tapahtuva AD-muuntaminen oli oletuksena aivan liian hidasta. Ohjelma ottaa oletuksena AD-muunnosta yhden millisekunnin välein ja tällä viiveellä päästään noin 1 kHz:n näytteenottotaajuuteen. Alun perin ohjelma käyttää kortin omaa sisäänrakennettua kiihtyvyysanturia ja tämä yhden kilohertsin näytteenottotaajuus sopii sille.

Lisäksi kyseinen ohjelma käyttää myös lämpötilamittausta. Lämpötilamittausta varten ohjelma käyttää FR5739-mikrokontrollerin FRAM-muistia. Kun tätä muistia käytetään, prosessorin kellotaajuuden on oltava maksimissaan 8 MHz. Näin matalalla kellotaajuudella ei voida tehdä kovin nopeaa AD-muuntamista. (Texas Instruments Processors Wiki 2012b.)

Koska tarkoituksena oli saada mahdollisimman nopeaa AD-muunnosta, ohjelmaan oli tehtävä muutoksia. Ohjelmasta poistettiin lämpötilanmittausfunktiot ja kellotaajuus nostettiin maksimiin eli 24 MHz:iin. Lisäksi ajastimen viive asetettiin mahdollisimman pieneksi. AD-muuntimen asetukset asetettiin myös erilailla. Alun perin ohjelma käytti

erilaisia virransäästöfunktioita. Jotta päästään tarpeeksi suuriin nopeuksiin, nämä virransäästöominaisuudet pitää poistaa.

15.2 Sisääntulopinnien asetukset

Datalogger-ohjelma käyttää oletuksena siis kiihtyvyysanturin asetuksia. Tässä työssä keskityttiin ensiksi enemmän AD-muuntamiseen, joten kiihtyvyysanturin sijaan käytettiin signaaligeneraattoria tuottamaan signaalia. Samat asetukset käyvät myös kiihtyvyysanturille.

Kiihtyvyysanturi tarvitsee toimiakseen kolme sisääntulopinniä kolmea akselia varten ja yhden virtapinnin anturin käyttöjännitettä varten. Kiihtyvyysanturissa on siis x, y ja z-akselit. Tässä työssä käytettiin MSP430-kortin 3.0, 3.1 ja 3.2 pinnejä sisääntuloina anturin akseleille. Anturin virtapinni on kortin pinni 2.7.
Anturin akselien pinnit määritellään seuraavalla tavalla:

```
#define   ACC_PORT_SEL0    P3SEL0
#define   ACC_PORT_SEL1    P3SEL1
#define   ACC_X_PIN        BIT0
#define   ACC_Y_PIN        BIT1
#define   ACC_Z_PIN        BIT2
```

Pinnit otetaan käyttöön seuraavalla tavalla:

```
ACC_PORT_SEL0 |= ACC_X_PIN + ACC_Y_PIN + ACC_Z_PIN;
ACC_PORT_SEL1 |= ACC_X_PIN + ACC_Y_PIN + ACC_Z_PIN;
```

Tässä jokainen kiihtyvyysanturin akselin sisääntulopinni kytketään toimintakuntoon.

Jos halutaan käyttää kortin sisäänrakennettua kiihtyvyysanturia, pitää määrittää myös kiihtyvyysanturin virtapinnit. Myös ulkoiselle kiihtyvyysanturille pitää määrittää virtapinnit, jos virta otetaan mikrokontrollerilta.

Virtapinnit määritetään seuraavalla tavalla:

```
#define   ACC_PWR_PIN        BIT7
#define   ACC_PWR_PORT_DIR   P2DIR
#define   ACC_PWR_PORT_OUT   P2OUT
```

Ja otetaan käyttöön:

ACC_PWR_PORT_DIR |= ACC_PWR_PIN;

ACC_PWR_PORT_OUT |= ACC_PWR_PIN;

15.3 AD-muuntimen asetukset

AD-muuntimelle täytyi asettaa oikeat asetukset, jotta saataisiin mahdollisimman nopeaa muunnosta. AD-muuntimelle täytyy määrittää sisääntulokanavat signaaleita varten seuraavalla tavalla:

```
#define   ACC_X_CHANNEL    ADC10INCH_12
#define   ACC_Y_CHANNEL    ADC10INCH_13
#define   ACC_Z_CHANNEL    ADC10INCH_14
```

AD-muuntimessa käytettiin seuraavia asetuksia:

ADC10CTL0 = ADC10SHT_2 + ADC10MSC + ADC10ON;

Tässä ADC10SHT_2 tarkoittaa näytteen pitoaikaa. Sillä määritetään kuinka monta kellojaksoa käytetään muunnoksen aikana. Numero kaksi tarkoittaa, että kellojaksoja on käytössä 16. ADC10MSC:lla kytketään moninkertainen näytteistäminen. ADC10ON-komennolla kytketään AD-muunnin toimintakuntoon.

ADC10CTL1 = ADC10SHP + ADC10CONSEQ_1;

Tässä kytketään pulssimuotoinen muunnos komennolla ADC10SHP. ADC10CONSEQ_1 tarkoittaa taas sitä, että käytetään useita kanavia.

ADC10CTL2 |= ADC10RES;

Tässä ADC10RES komennolla kytketään 10 bittinen resoluutio.

ADC10MCTL0 = ADC10INCH_14;

Tässä kytketään käyttöön kaikki 14 kanavaa.

Näillä asetuksilla pitäisi päästä 20 kHz:n näytteenottotaajuuteen, mutta muista asetuksista riippuen muunnosnopeus on paljon enemmän. Varsinainen AD-muunnos

tapahtuu ikisilmukassa ja muunnokset otetaan talteen jokaiselta kanavalta erikseen ADC_Result-taulukkoon.

15.4 DMA-asetukset

Jotta AD-muunnos saadaan tarpeeksi nopeaksi, tarvitaan avuksi MSP430-mikrokontrollerin DMA-ohjainta.

Tässä ohjelmassa käytettiin seuraavia asetuksia DMA:lla:

DMACTL0 = DMA0TSEL__ADC10IFG;

Tässä DMACTL0-rekisteriin asetetaan DMA:ssa toimiva AD-muuntimen ADC10IFG liipaisin päälle.

DMA0CTL = DMADT_4 + DMADSTINCR_3 + DMAEN + DMAIE

DMA:n kanavan DMA0CTL-rekisteriin asetetaan DMA:n siirtomuoto ja koko DMA toimimaan. Lisäksi asetetaan DMA:n keskeytyspalvelu.

Komennolla __data16_write_addr((unsigned short) &DMA0SA,(unsigned long) &ADC10MEM0); AD-muuntimen tallennusrekisterin tiedot siirretään DMA:n muistiin. Tämä on niin sanotusti DMA:n lähdeosoite.

__data16_write_addr((unsigned short) &DMA0DA,(unsigned long) &ADC_Result[0]); on komento, jolla ADC10MEM0:n tiedot siirretään ADC_Result nimiseen taulukkoon. Tällä komennolla siis osoitetaan osoitteen määränpää.

Tässä työssä DMA:ta käytettiin keskeytysperiaatteen mukaan. Aina kun AD-muunnos on valmis, muunnettu data siirretään DMA:n muistiin.

15.5 Ajastimen asetukset

Ohjelmassa käytettiin ajastinta AD-muunnoksen ottamista varten. Ajastimen avulla muunnosta voidaan ottaa jatkuvasti erillisessä funktiossa ilman muun ohjelman vaikutusta.

Ajastimen asetukset ovat seuraavat:

TA0CCTL0 = CCIE;

Tässä asetetaan ajastimen keskeytys päälle.

TA0CCR0 = 130;

Tässä asetetaan aika.

TA0CTL = TASSEL_2 + MC_2;

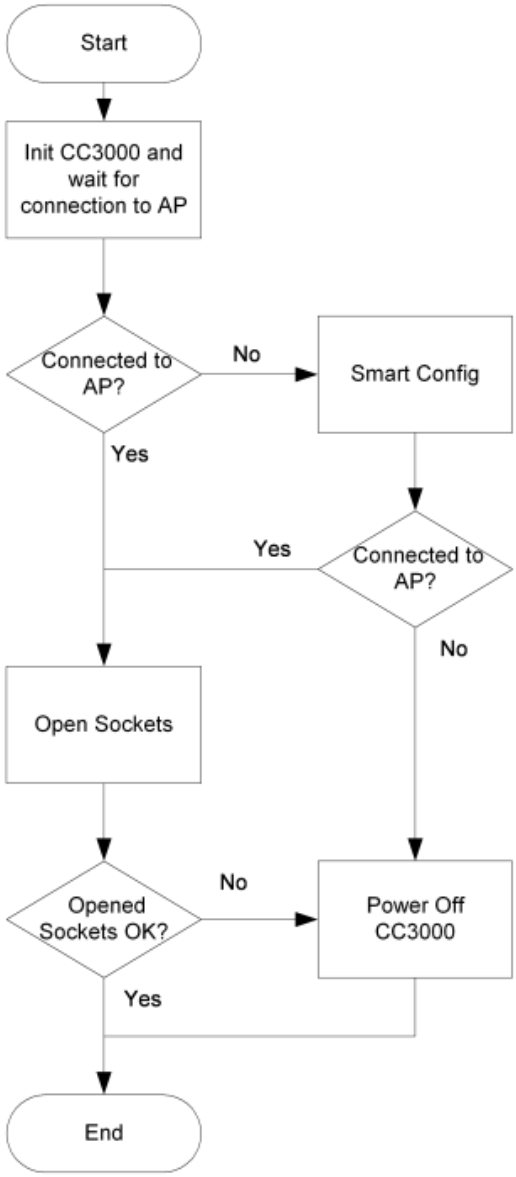
Tässä kytketään ajastimen kellosignaali ja jatkuva tila päälle.

Ajastin pyörii erillisessä keskeytyksellä toimivassa funktiossa, jossa samassa kutsutaan AD-muunnoksen funktiota.

16 WLAN-LÄHETYKSEN OHJELMA

Työssä käytettiin pohjana Texas Instrumentsin tekemää WLAN-ohjelmaa. WLAN-ohjelma sisältää paljon erilaisia alustustiedostoja, joita WLAN-moduuli tarvitsee toimiakseen. Nämä tiedostot on mahdotonta tehdä itse, joten järkevintä on käyttää valmista pohjaa. WLAN-ohjelma käyttää tiedonsiirtoon UDP-protokollaa, joka soveltuu hyvin tällaiseen käyttöön, koska jatkuvaa yhteyttä ei tarvita. Ohjelma sisältää CC3000 WLAN -moduulin ohjaukseen tarvittavat tiedostot. Datalogger-ohjelma sisältää WLAN-ohjelman sekä AD-muuntamisen funktiot. MSP430FR5739-mikroprosessorin pääohjelma on liitteessä 3 ja AD-muunnoksen ohjelma liitteessä 4. Näiden lisäksi ohjelmassa on monia muita tiedostoja, joita WLAN-moduuli tarvitsee toimiakseen.

Kuviossa 16 on esitetty WLAN-ohjelman yhteydenmuodostus. WLAN-moduuli kytketään toimimaan CC3000EnableAndOpenPort-funktiolla. Tässä funktiossa alustetaan WLAN-moduulin ajurit ja muodostetaan yhteys tukiasemaan. Kun yhteys on saatu, avataan portit. Jos yhteyttä ei jostain syystä saada, ohjelma suorittaa ns. Smart Config -funktion, joka asettaa automaattisesti oikeat asetukset. Jos tämänkään jälkeen ei saada yhteyttä tukiasemaan, WLAN-moduuli sammutetaan.



KUVIO 16. WLAN-ohjelman yhteyden muodostaminen (Texas Instruments Processors Wiki 2012b)

Ohjelman pääfunktiossa alustetaan kaikki liitännät ja tarvittavat funktiot. Pääohjelmassa olevassa ikisilmukassa toteutetaan WLAN-moduulin kytkeminen ja yhteyden muodostus sekä datan lähetys. Ohjelmassa voidaan valita, käytetäänkö suojattua WLAN-yhteyttä vai ei. Tässä tapauksessa käytettiin suojaamatonta yhteyttä, jotta yhteys toimisi varmasti.

WLAN-ohjelmaan liitetty AD-muunnin-funktio ottaa jatkuvaa AD-muunnosta ja saatu data lähetetään WLAN:n avulla eteenpäin. Erillinen lähetysfunktio kytkeytyy toimintakuntoon, kun WLAN-yhteys muodostetaan. Lähetysfunktio lähettää 7 tavun paketteja, joista yksi tavu kertoo datan tyypin ja kuusi seuraavaa kiihtyvyysanturilta saadut tiedot. Jokaiselle kiihtyvyysanturin akselille on varattu kaksi tavua.

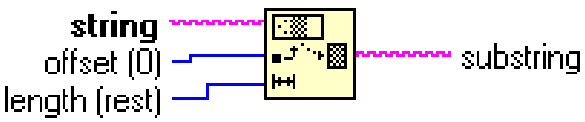
Ohjelmaan tehtiin muutamia muutoksia, jotta siitä saataisiin yksinkertaisempi. Alun perin AD-muunnoksen lähetys tapahtui, kun painettiin MSP430-kortin painiketta. Testit tehtiin ilman tätä painiketta ja painikkeen kanssa. Ohjelma sisälsi oletuksena paljon erilaisia viiveitä, jotka hidastivat WLAN-lähetystä jonkin verran, joten nämä viiveet poistettiin ohjelmasta.

17 DATANKERÄYSOHJELMA

Texas Instrumentsin datankeräysohjelman mukana tulee Windowsille ohjelma, joka kerää dataa langattomasti MSP-kortilta. Tässä työssä ei käytetty kyseistä valmista ratkaisua, vaan tehtiin oma datankeräysohjelma LabView-ohjelmistoympäristössä.

LabView-ohjelmasta tehtiin mahdollisimman yksinkertainen datankeräysohjelma. Ohjelma lukee UDP-protokollaa ja muuntaa saapuvan datan oikeaan muotoon. UDP:stä tuleva data on jonossa, ja jonosta täytyy poimia erilleen kiihtyvyysanturin akselien tiedot. Ensiksi lähetettiin vain yhden akselin tietoa eli yhden kanavan dataa. Sitten kun ohjelman toiminta voitiin varmistaa, tehtiin tarvittavat muutokset, jotta voitiin lukea dataa kolmelta kanavalta.

Vastaanotetun jonon koko on siis 7 tavua ja tavut 1 – 2 on x-akselin data. Y-akselin data on tavut 3 – 4 ja z-akselin data on loput tavut eli 5 – 6. Ensimmäinen tavu eli tavu nolla on lähetettävän datan tieto. Jonosta poimiminen toteutettiin LabView-ohjelman String subset -funktiolla (kuvio 17). Tähän funktioon asetetaan haluttu jonon kohta, joka siirretään eteenpäin.



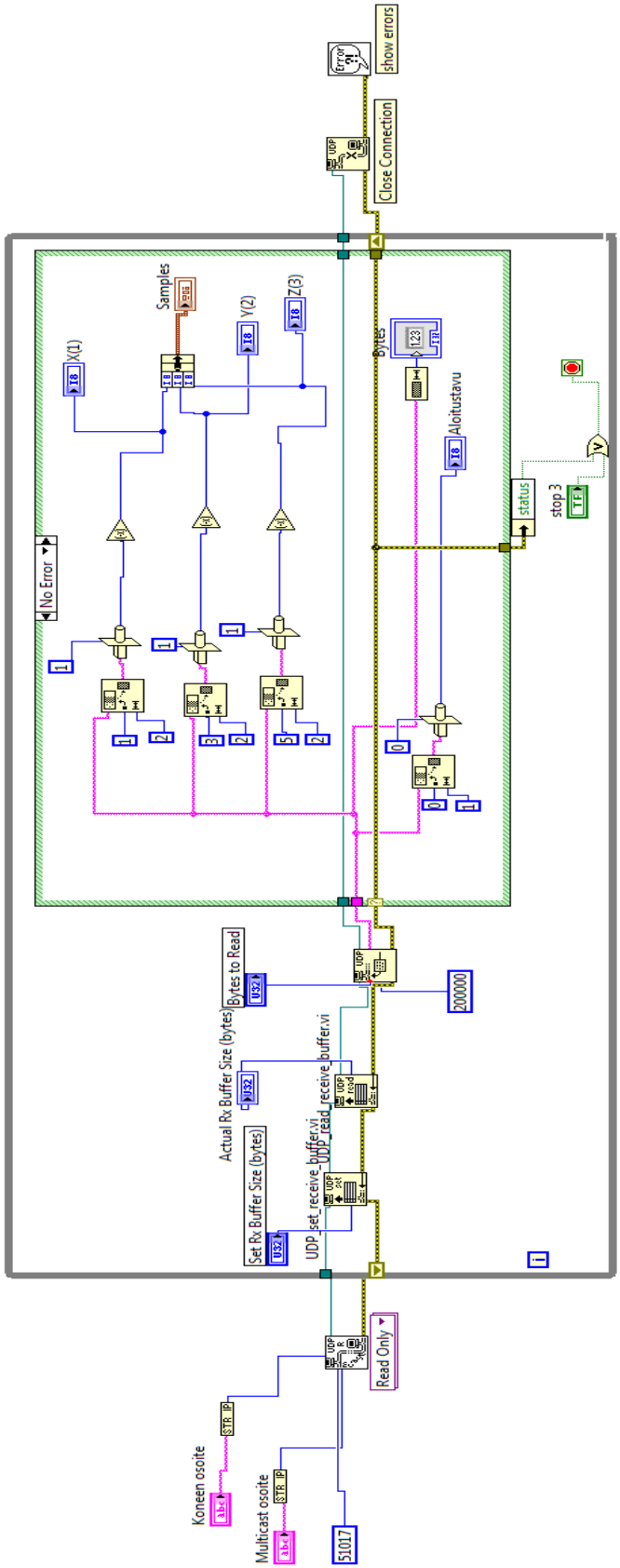
KUVIO 17. String subset - funktio

Kun jonosta on eroteltu saapuva data, se muunnetaan oikeaan muotoon. Tässä tapauksessa käytettiin 8-bittistä integer-muotoa, mutta 16-bittistä muotoa voisi käyttää. Muunnettu data esitetään sitten ohjelman pääikkunan kuvaajassa.

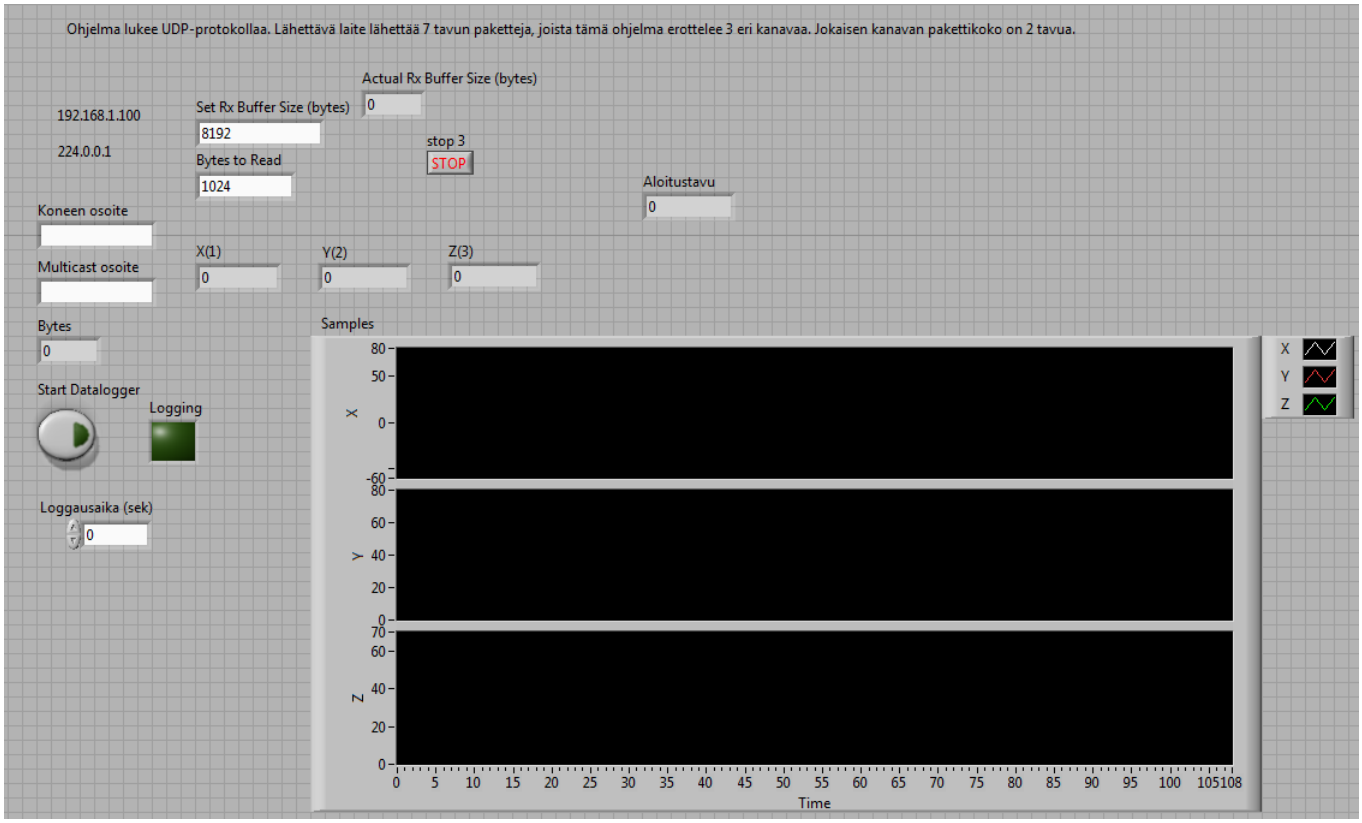
LabView-ohjelmaan sisällytettiin myös datantallennustoiminto. Data tallennetaan kolmesta eri lähteestä tekstitiedostoon. Tallennusaika voidaan säätää käsin ohjelman

pääikkunassa. Data voidaan tallentaa myös muihin tiedostomuotoihin, kuten Excel-tiedostoon. Kuviossa 18 on esillä LabView-ohjelman lohkokaavio ja kuvassa 19 ohjelman käyttöikkuna.

Ohjelmaan lisättiin varmuuden vuoksi UDP:n vastaanottopuskurin kasvattamista varten pienet aliohjelmat. Vastaanottopuskurin kasvattaminen ei ole välttämätöntä, mutta jossain tilanteissa voidaan tarvita lisää puskuria. Vastaanottopuskurin kooksi asetettiin 8 192 tavua.



KUVIO 18. LabView-ohjelman lohkokaavio

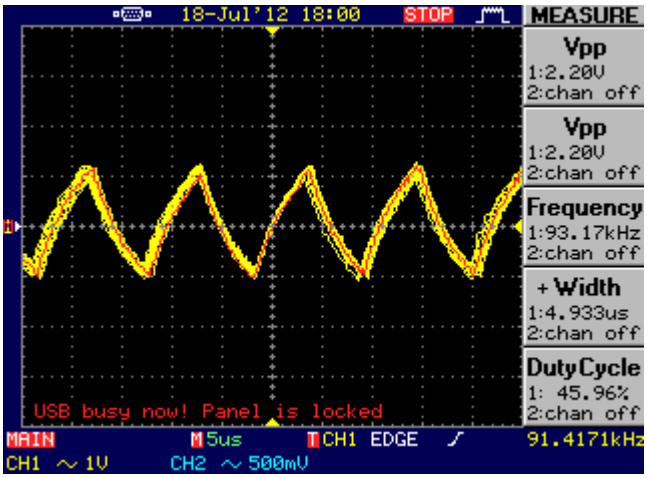


KUVA 19. LabView-ohjelman pääikkuna

Pääikkuna toimii seuraavalla tavalla: Osoitekenttiin syötetään tietokoneen IP-osoite ja Multicast-osoite, joka tässä tapauksessa on 224.0.0.1. Portin numero saatiin selville MSP430-kortin ohjelmasta. Portti asetetaan ohjelman lohkokaavioon. Lisäksi pääikkunassa on datankeräyspainike, josta alkaa datan talteenotto. Kuvaajassa näkyy x, y ja z-akselit. Lisäksi akselien arvot näkyvät myös numeroina.

18 AD-MUUNNOKSEN LÄHETYKSEN TESTAUS

Lukuisten testien ja asetusten vaihtojen jälkeen päästiin AD-muuntamisessa tyydyttävään lopputulokseen. AD-muuntaminen toimii siis ajastimen avulla. Eri ajastinarvoilla kokeilemalla selvisi, että suurin toimiva muunnosnopeus on n. 90 Ksps. Jos tämän ylittää, WLAN-lähetys ei toimi. Voi olla, että joillakin asetuksilla WLAN-lähetys toimisi nopeamman muunnoksen kanssa. Kuvassa 20 on muunnosnopeus oskilloskoopilta, kun muunnetaan signaalia kolmelta kanavalta ja WLAN-lähetys toimii.



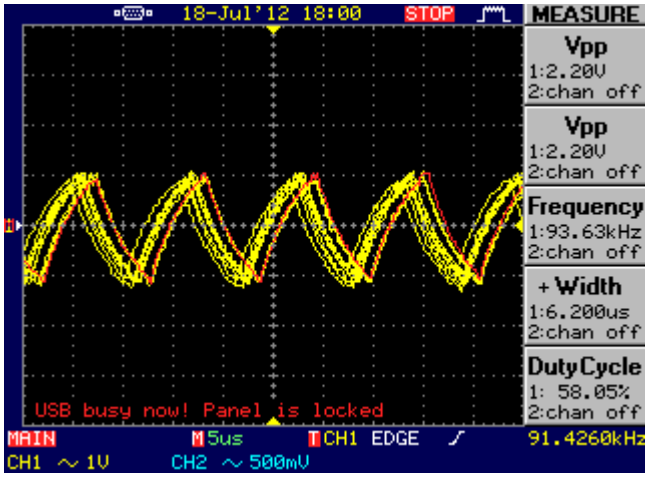
KUVA 20. Muunnosnopeus oskilloskoopilta

WLAN-muuntimen funktioon täytyi tehdä joitain muunnoksia, jotta päästäisiin kuvan 15 muunnosnopeuteen. Esimerkiksi ohjelmasta piti poistaa silmukka, joka kertoo, onko muunnos valmis. Tämä while-silmukka on:

while (ADC10CTL1 & BUSY)

BUSY-komento indikoi, onko AD-muunnin valmis. Tämä silmukka aiheutti suuria viiveitä ja hidasti AD-muuntamista. Lisäksi DMA:n käyttämä virransäästöominaisuus piti ottaa pois käytöstä.

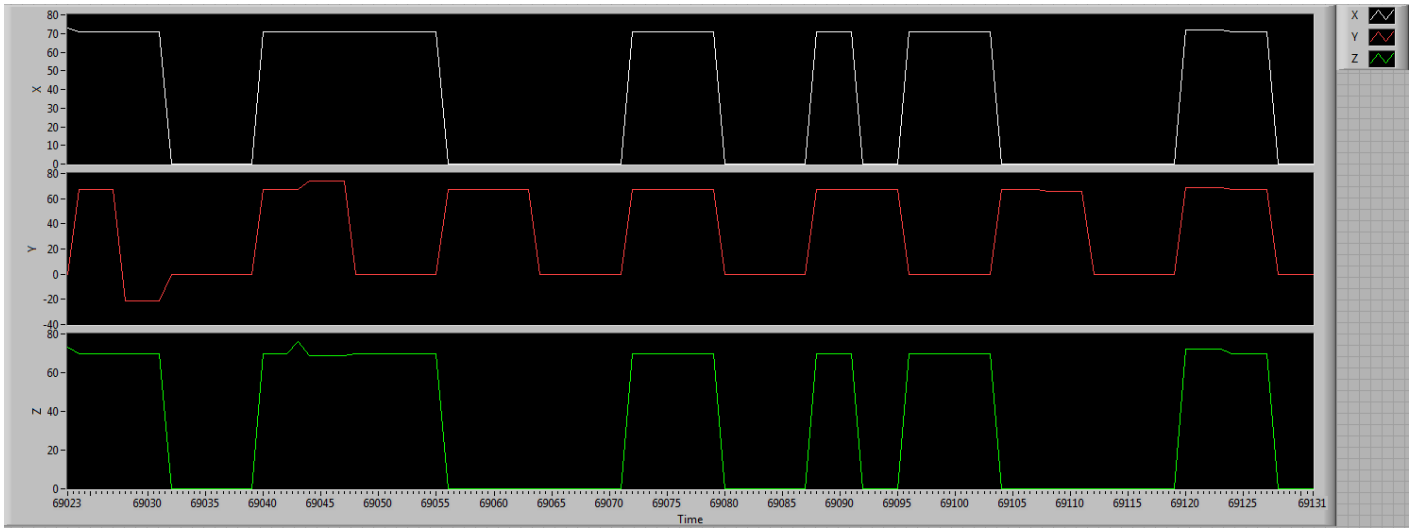
Testien perusteella todettiin, että tämä mikrokontrolleri ei välttämättä sovellu näin nopeaan AD-muuntamiseen ja datan lähetykseen. Suurilla muunnosnopeuksilla WLAN-lähetys alkaa vaikuttaa muunnosnopeuteen. Oskilloskoopista huomaa, kuinka signaali alkaa väristä, kun kytkee lähetyksen. Oletusnopeuksilla tätä värinää ei tapahdu. Kuvassa 21 on muunnosnopeus, kun WLAN-lähetys on kytketty toimimaan.



KUVA 21. Muunnosnopeus oskilloskoopilta ja WLAN-lähetys

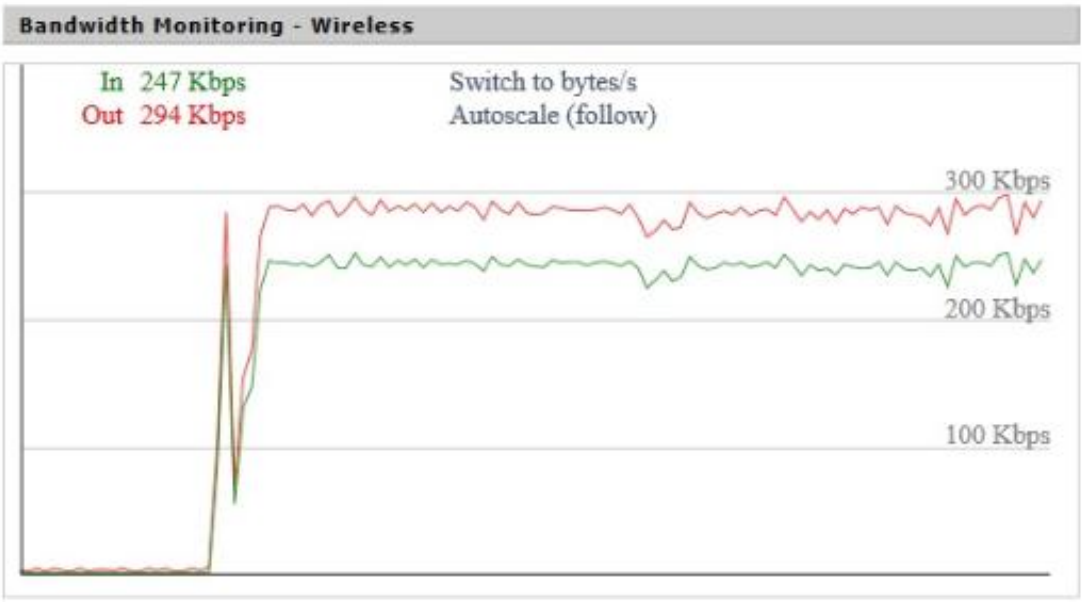
Muunnosnopeus niin sanotusti pätkii todella nopeasti, kun WLAN-lähetys on käytössä. Tähän ei ole löytynyt selvyyttä, mistä se johtuisi. Pätkiminen voi johtua monesta eri asiasta.

Data liikkuu nopeasti LabView-ohjelmaan, vaikka muunnos vähän pätkiikin. AD-muunnetun datan lähetys toimii siis riittävällä tavalla. Kuvassa 22 on LabView-ohjelmaan lähetetty signaali. Valkoinen signaali on x-kanavan signaali ja punainen y-kanavan. Vihreän z-kanavan signaali on periaatteessa sama kuin x-kanavan, koska molempien kanavien lähdesignaali on samasta funktiogeneraattorista. Kaikille kanaville ajettiin 200 Hz:n kanttiaaltoa, joka AD-muunnettiin.



KUVA 22. WLAN-signaali LabView-ohjelmassa

AD-muunnoksen lähetysnopeuden kaistanleveydeksi saatiin n. 300 Kbps. Data siis liikkui WLAN:ssa n. 300 kilotavua sekunnissa. Kaistanleveyden koon pystyi tarkastamaan WLAN-reitittimen seurantaohjelmasta. Kuvassa 23 on esillä lähetyksen kaistanleveys.

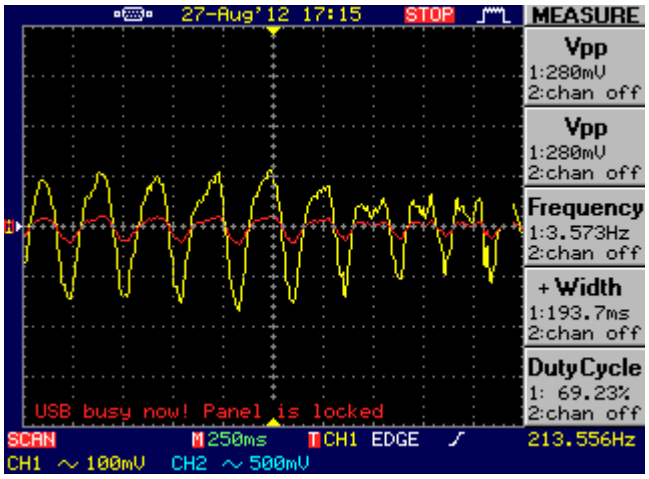


KUVA 23. WLAN – lähetyksen kaistanleveys

19 VÄRÄHTELYN MITTAUKSEN TESTAUS JA DATAN LÄHETYS

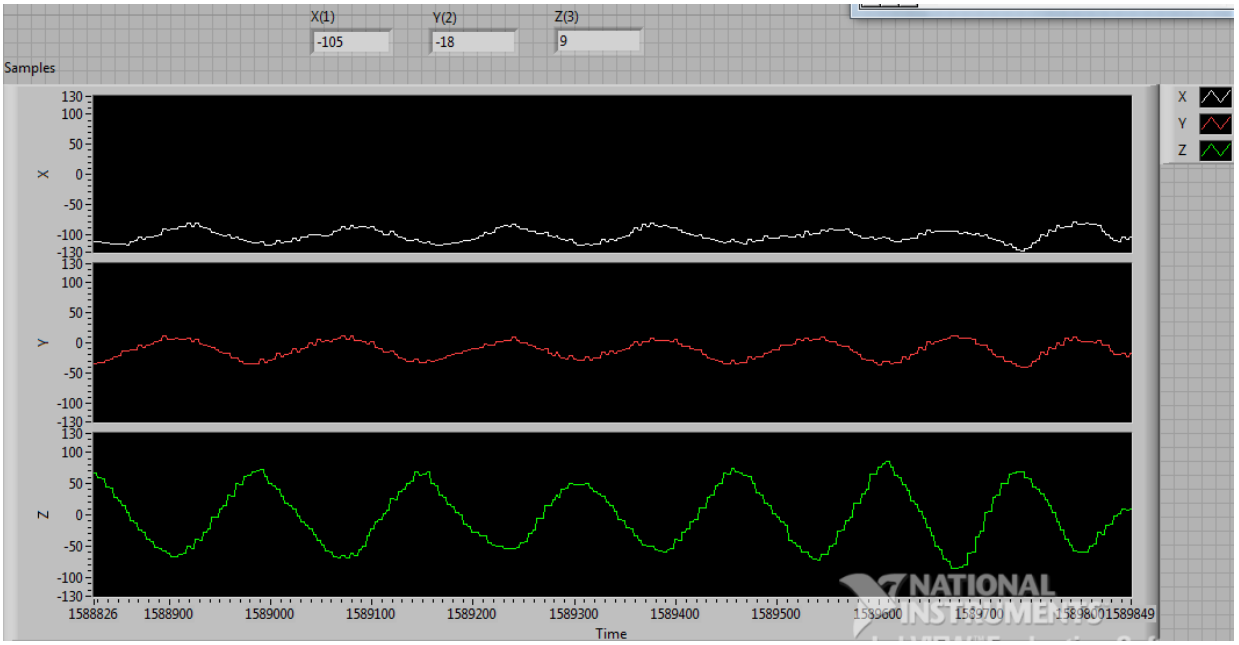
Työn lopullisena tarkoituksena oli saada mitattua kiihtyvyysanturilla värähtelyä ja lähettää mitattu data langattomasti tietokoneelle. MSP430FR5739-mikrokontrollerissa on sisäänrakennettuna kiihtyvyysanturi, mutta se ei sovellu kovin hyvin värähtelyn mittaukseen, koska se on kortissa kiinni. Kiihtyvyysanturin kanssa voidaan käyttää melkein samaa ohjelmaa, jota käytettiin AD-muunnoksen lähettämiseen. Mikrokontrollerin käsittelemä kiihtyvyysanturin data on raakaa käsittelemätöntä dataa. Jos halutaan oikeanlaista dataa, ohjelmaan on lisättävä AD-muunnoksen lisäksi signaalinkäsittelyä.

Kiihtyvyysanturin toiminnan MSP430-kortin kanssa pystyi testaamaan kortin omalla kiihtyvyysanturilla. Ensiksi kiihtyvyysanturin signaalit testattiin oskilloskoopilla. Kuvassa 24 on kiihtyvyysanturin signaalia suoraan anturista, kun anturi altistuu tärinälle.

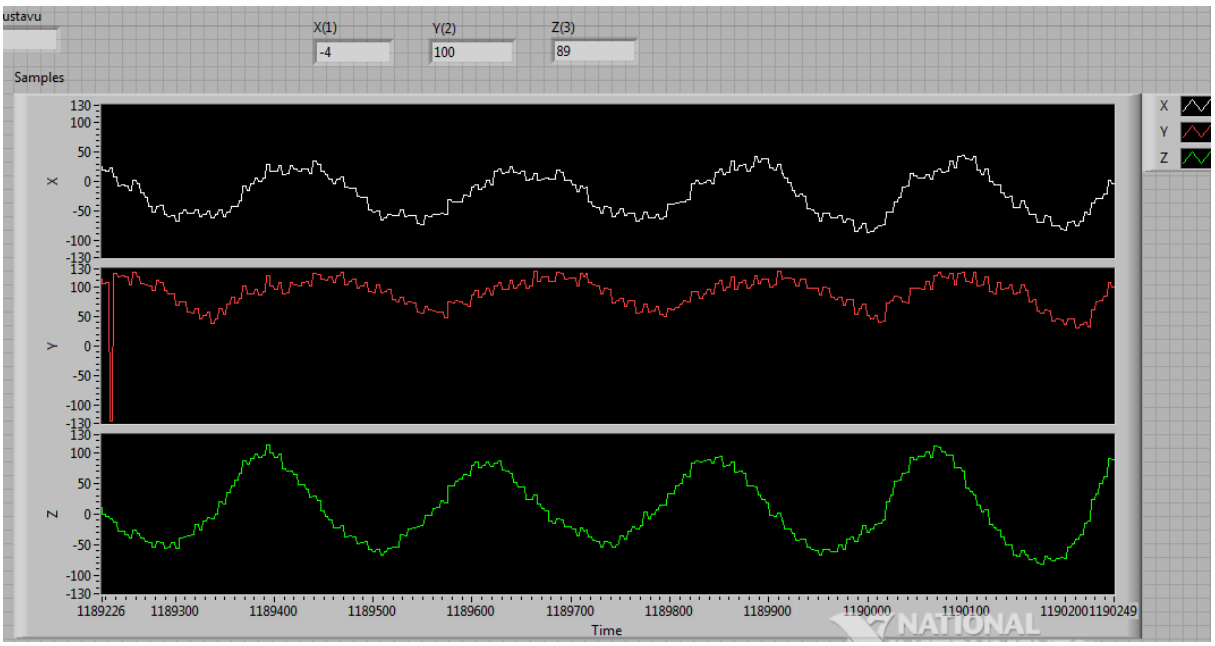


KUVA 24. Kiihtyvyysanturin signaalia

Tämän jälkeen testattiin kiihtyvyysanturin AD-muunnetun datan lähetystä langattomasti. MSP-kortille aiheutettiin tärinää, joka vastaa värähtelyn mittausta. Testien perusteella data liikkui samalla tavalla LabView-ohjelmaan kuin aikaisemmin pelkän AD-muunnoksen lähetyksessä. Kuvissa 25 ja 26 on sisäänrakennetun kiihtyvyysanturin signaalia LabView-ohjelmassa.



KUVA 25. Sisäänrakennetun kiihtyvyysanturin testaus



KUVA 26. Sisäänrakennetun kiihtyvyysanturin testaus

KUVA 27. Kovaa tärinää sisäänrakennetulla anturilla

Kuvassa 25 värähtely aiheutuu eniten sensorin z-akselille ja kuvassa 26 värähtelyä on havaittavissa jokaisella akselilla. Ensimmäinen arvo on x-akselilta ja toinen arvo on y-akselilta. Viimeinen arvo on z-akselin arvo. Kuvassa 27 on testitulos, kun anturille tehdään kovaa tärinää. Kuten kuvista näkee, signaaleissa on jonkin verran kulmikkuutta. Signaali ei siis ole puhdasta siniaaltoa, jota sen pitäisi olla. värähtelytestien yhteydessä testattiin datankeräyksen toiminta ja värähtelyn arvoja tallennettiin datankeräysohjelmalla tekstitiedostoon (kuvio 28).

```
27.8.2012,12:19:29          36,   -32,   88
27.8.2012,12:19:29          36,   -32,   88
27.8.2012,12:19:29          36,   -32,   88
27.8.2012,12:19:29          36,   -32,   88
27.8.2012,12:19:29          36,   -32,   88
27.8.2012,12:19:29          61,   -37,   85
27.8.2012,12:19:29          61,   -37,   85
27.8.2012,12:19:29          61,   -37,   85
```

KUVIO 28. Otos datankeräys-tekstitiedostosta

20 YHTEENVETO

Opinnäytetyön tavoitteena oli kehittää mittauslaitteisto, jolla voidaan mitata voimaa ja värähtelyä. Lisäksi mitattu data tuli saada siirrettyä langattomasti tietokoneelle. Lopputuloksena saatiin toimiva datankeräysjärjestelmä kiihtyvyysanturille. Kiihtyvyysanturi saatiin toimimaan prosessorikortin kanssa ja mitattu data saatiin lähetettyä langattomasti tietokoneelle datankeräysohjelmaan. Lisäksi venymäliuskaprosessori saatiin toimimaan halutulla tavalla. Työn lopputulokseen ollaan kaikin puolin tyytyväisiä.

Työ sisälsi mikrokontrollerin ohjelmointia C- ja Assembler-kielillä sekä LabView-ohjelmointia. Lisäksi työ sisälsi sensoritekniikkaa, sulautettujen järjestelmien kehitystä ja langatonta tiedonsiirtoa. Työ oli siis erittäin laaja ja opin paljon uutta sulautettujen järjestelmien ohjelmoinnista ja sensoritekniikasta. Työstä saatuja oppeja voin hyödyntää tulevaisuuden töissä.

LÄHTEET

Acam. 2012a. *PS081-EVA-Kit datasheet.* [Pdf-dokumentti]. [viitattu 8.5.2012]. Saatavissa: http://www.acam.de/fileadmin/Download/pdf/English/DB_PS081-EVA_en.pdf

Acam. 2012b. *PSØ81 datasheet.* [Pdf-dokumentti]. [viitattu 9.5.2012]. Saatavissa: http://www.acam.de/fileadmin/Download/pdf/English/DB_PS081_en.pdf

Analog Devices. 2012. *AD8420 Datasheet.* [Pdf-dokumentti]. [viitattu 20.6.2012]. Saatavissa: http://www.analog.com/static/imported-files/data_sheets/AD8420.pdf

Caro, D. 2008. *Wireless Networks for Industrial Automation (3rd Edition).* [verkkokirja]. ISA. [viitattu 25.7]. Saatavilla: http://www.knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=3309&VerticalID=0

Chen, W-K. 2005. *Electrical Engineering Handbook* [verkkokirja]. Elsevier. [viitattu 11.10]. Saatavilla: http://www.knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=1713&VerticalID=0

Gardner, J.W., Varadan, V.K., Awadelkarim, O.O. 2001. *Microsensors, MEMS, and Smart Devices.*[verkkokirja]. John Wiley & Sons. [viitattu 31.7.2012]. Saatavissa: http://www.knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=1436&VerticalID=0

Gastineu, A., Johnson, T. & Schultz, A. 2009. *Bridge Health Monitoring and Inspection – A Survey of Methods.* [Pdf-dokumentti]. [viitattu 12.9.2012]. Saatavissa: http://www.cts.umn.edu/Publications/ResearchReports/pdfdownload.pl?id=1262

Jing, C., Chun, W., Chun, X. & Liqiao, L. 2009. *Research on Signal Processing for Bridge Health Monitoring.* [Pdf-dokumentti]. [viitattu 12.9.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=5274059

Kyowa. 2012. *Strain Gages*. [Pdf-dokumentti]. [viitattu 14.6.2012 ja 19.6.2012]. Saatavissa: http://www.straintech.fi/pdf/2012Kyowastraingage_cat_e.pdf

Lidan, X., Yin-Nee, C., Chen, C., Mo, Y., King-Lun, K., Po-Fat, C., Leung, W. 2007. *A Strain Gauge that Uses Carbon Black and Carbon Nanotube Doped Silicone Oil Encapsulated in a PDMS Microchannel.* [Pdf-dokumentti]. [viitattu 12.6.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=4601398

LS Research. 2012. *TiWi-SL Module – Datasheet.* [Pdf-dokumentti]. [viitattu 19.7.2012]. Saatavissa: http://www.lsr.com/downloads/products/330-0085.pdf

Mhetre, M. Nagdeo, N.S. & Abhyankar, H. K. 2011. *Micro Energy Harvesting for Biomedical Applications: A Review.* [Pdf-dokumentti]. [viitattu 20.6.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=5941789

Millau Viaduct. 2012. *Techniques: Instrumentation*. [verkkodokumentti]. [viitattu 11.9.2012]. Saatavissa: http://www.leviaducdemillau.com/en_index.php#/accueil/

Ming, L., Xiaokun, S. & Yawen, W. 2008. *Signal Processing and Accelerometer-based Design for Portable Small Displacement Measurement Device.* [Pdf-dokumentti]. [Viitattu 31.7.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=4595612

Morris, A. S. & Langari, R. 2012. *Measurement and Instrumentation - Theory and Application.* [verkkokirja]. Elsevier. [viitattu 11.10]. Saatavilla: http://www.knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=4676&VerticalID=0

Pararas-Carayannis, C**.** 2007. *Rio–Antirrio bridge*. [verkkodokumentti]. [Viitattu 11.9.2012]. Saatavissa: http://www.drgeorgepc.com/ArtRionAntirionBridge.html

PCB Piezotronics. 2012. *Sensing Technologies used for Accelerometers.* [verkkodokumentti]. [viitattu 9.10]. Saatavissa: http://www.pcb.com/Accelerometers/Sensing_Technologies.asp

Rogers, J.E., Ramadoss, R., Ozmun, P.M., & Dean, R.N. 2007. *MEMS Accelerometer Fabricated Using Printed Circuit Processing Techniques.* [Pdf-dokumentti]. [viitattu 20.6]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=4375135

Spectrum.ieee. 2012. *Despite Stimulus Money, Most U.S. Bridges Might Stay Dumb.* [verkkodokumentti]. [viitattu 11.9.2012]. Saatavissa: http://spectrum.ieee.org/computing/hardware/despite-stimulus-money-most-us-bridges-might-stay-dumb/0

Stefanescu, D.M. 2011. *Strain gauges and Wheatstone bridges — Basic instrumentation and new applications for electrical measurement of non-electrical quantities*. [Pdf-dokumentti]. [viitattu 7.6.2012 ja 14.6.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=5767428

Texas Instruments. 2012a. *MSP430 Ultra-Low-Power-Microcontrollers.* [Pdf-dokumentti]. [viitattu 18.7.2012]. Saatavissa: http://www.ti.com/lit/sg/slab034v/slab034v.pdf

Texas Instruments. 2012b. MSP430FR5739. [verkkodokumentti]. [viitattu 18.7.2012]. Saatavissa: http://www.ti.com/tool/msp-exp430fr5739

Texas Instruments. 2012c. *MSP430FR573x, MSP430FR572x Mixed Signal Microcontroller.* [Pdf-dokumentti]. [viitattu 16.10]. Saatavissa: http://www.ti.com/lit/ds/symlink/msp430fr5739.pdf

Texas Instruments. 2012d. *MSP430FR5739 – Datasheet.* [Pdf-dokumentti]. [viitattu 17.5.2012 ja 18.7.2012]. Saatavissa: http://www.ti.com/lit/ug/slau272a/slau272a.pdf

Texas Instruments. 2012e. *Energy Harvesting.* [verkkodokumentti]. [viitattu 12.10.2011]. Saatavissa: http://www.ti.com/ww/en/apps/energy-harvesting/index.shtml?DCMP=MSP430_Energy&HQS=Other+OT+energyharvesting

Texas Instruments Processors Wiki. 2012a. *TiWi-SL Module*. [verkkodokumentti]. [viitattu 19.7]. Saatavissa: http://processors.wiki.ti.com/index.php/CC3000_Wi-Fi_EM#LS_Research_EM_Board

Texas Instruments Processors Wiki. 2012b. *CC3000 Data Logger.* [verkkodokumentti]. [viitattu 10.8]. Saatavissa: http://processors.wiki.ti.com/index.php/CC3000_Data_Logger_Description

Wilson, J. S. 2005. *Sensor Technology Handbook* [verkkokirja]. Elsevier. [viitattu 20.6.2012]. Saatavilla: http://www.knovel.com/web/portal/browse/display?_EXT_KNOVEL_DISPLAY_bookid=1659&VerticalID=0

Yongdae, K. Youngdeok, K., Chulsub, L. & Sejin, K. 2010.*Thin Polysilicon Gauge for Strain Measurement of Structural Element.* [Pdf-dokumentti]. [viitattu 7.6.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=5471796

Zuozhou, Z., Jiangbo, S. Xiaotian, F. Wei, L. Xiaohui, C. Zonggang, W. & Huazhong, Y. 2012. *Wireless Sensor Network Based Cable Tension Monitoring for Cable-stayed Bridges*. [Pdf-dokumentti]. [viitattu 12.9.2012]. Saatavissa: http://ieeexplore.ieee.org.ezproxy.savonia-amk.fi:2048/stamp/stamp.jsp?tp=&arnumber=6174723

ESIMERKKI VOIMANMITTAUSOHJELMA PS081-KORTILLA

```
;Example program
;Author: Acam messelectronic, UTG / RE
Date: 01-06-2008


#include "config.h"          ;include the configuration of the registers, please see 'Includes' tab
page

;------------LCD register  ----------------------------------------------------
CONST lcd_reg_high      15 + 48
CONST lcd_reg_mid       14 + 48
CONST lcd_reg_low       13 + 48
;-------------------------------------------------------------------------------

;--------------- constants for RAM addresses in which values are stored-------------------------------------
----
CONST       init_offset_value           1
CONST       unit_mode                           2
CONST       state_machine                       3
CONST       state_measure                       4
CONST       counter                                     5
CONST       last_display_value          6
CONST roll_avg_ram_1    7
CONST roll_avg_ram_2    8
CONST roll_avg_ram_3    9
CONST roll_avg_ram_4    10
CONST roll_avg_result               11                  ;rolling average result
CONST       tara_value                          12

;-------------------------------------------------------------------------------------------------------------------------------
--

;--------------- constants for the states of the state machine-----------------------------------------------
CONST       idle_state                  0
CONST init_state                        1
CONST measure_state     2
```

```
;-------------------------------------------------------------------------------------------------------------------------
--

;--------------- constants for RAM addresses used by the TDC-----------------------------------------------------
-----
CONST       status_flags                22
CONST       result_HB0                  20
;-------------------------------------------------------------------------------------------------------------------------
---

;--------------- constants for bits in the flag register at RAM address 22 (status_flags)----------------------
------
CONST flag_Cal_button                   20          ; Cal button pressed
CONST       flag_POR                                        19          ;       EEpromein-
sprung durch Poweron Reset
CONST       flag_rstssn                                     18          ;       EEpromein-
sprung durch Button
CONST       ON_button_rise_flag         4           ; Rising edge flag for ON / Tara button
;-------------------------------------------------------------------------------------------------------------------------
----

;--------------- constants for configuration registers and configuration bits-----------------------------------
-------
CONST cfg_reg_lcd_standby                   59
CONST cfg_bit_lcd_standby                   12

CONST cfg_reg_tdc_sleepmode         49
CONST cfg_bit_tdc_sleepmode                 17
;-------------------------------------------------------------------------------------------------------------------------
-----

; Overview states:

; STATE == 1   --> init state
; STATE == 2   --> measure mode

;-------------------------------------------------------------------------------------------------------------------------
-----
```

```
; Power On Reset Detection

;-------------------------------------------------------------------------------------------------------------------------
-----

rst_pwr_check:
        ramadr      status_flags
        gotoBitS    r, flag_POR, power_on_reset   ; Reset occured let's go and find out
which one

program_entry:
        ramadr      state_machine           ; Normal program entry (no POR)

        getflag     r                       ; If state is zero we are in sleep mode


        gotoBitS    r, 0, get_initial_offset    ; if bit 0 is set it indicates init state

        gotoBitS    r, 1, measurement       ; if bit 1 is set it indicates measure state



power_on_reset:
        skipBitS    r, flag_rstssn, 2       ; Keep unit setting, if reset is from button
        ramadr      unit_mode               ; if new battery set unit to default
        clear       r                       ; Default unit mode is gr (0)

        ramadr      state_machine           ;After reset set state machine to sleep mode
(0)
        clear       r
        ramadr      counter                 ;initialize counter variable to zero
        clear       r


init:
        ramadr      state_machine
        move        r, 1                    ; Set state machine to initialization state (1)
```

```
            ramadr      last_display_value      ; This value holds the last value shown on
the display
            clear       r                       ; Nothing displayed yet so set to zero
            ramadr      tara_value
            clear       r                       ;clear saved tara
            ramadr      counter
            clear       r

            setLCD                              ; set all segments on LCD
            newlcd                              ; refresh LCD display

            clear       x
            jsub        roll_avg_initialize     ; Initialize rolling average with 0

;-------------------------------------------------------------------------------------------------------------------------
------------------
; The following bits are located in the configuration registers. By default the configuration should be
set up for sleep mode in which the current consumption is low.
; See the Configuration file in the include section. Only after turning on the scale they are changed
here to perform the desired operation.
;-------------------------------------------------------------------------------------------------------------------------
------------------
            ramadr      cfg_reg_lcd_standby     ; Register 59 contains config bit to put LCD
standby
            bitclr      r, cfg_bit_lcd_standby  ; Clear bit 12 to take LCD out of standby
(sleep) mode

            ramadr      cfg_reg_tdc_sleepmode   ;Register 49 contains config bit to put TDC
to sleepmode
            bitclr      r, cfg_bit_tdc_sleepmode ;Clear bit 17 to take TDC out of sleep mode
;-------------------------------------------------------------------------------------------------------------------------
--------------------
            goto        end

get_initial_offset:
            ramadr      result_HB0              ; Get measurement value for compensated
HB0 result
            move        x, r                    ; Move value to accu x
```

```
        jsub    five_times_roll_avg     ; Add value of x to rolling average 5 (result is
also stored in x)
        ramadr  init_offset_value       ; This location holds the value of the initial
offset
        move    r, x                    ; Averaged value is returned in x, save it as
initial offset
        ramadr  counter
        incr    r
        compare r, 10
        skipPos 3
        ramadr  state_machine           ; Next step sets state machine to measure
mode (2)
        move    r, 2
        jsub    roll_avg_initialize     ; Initialize rolling average with new offset
value


        goto    end

measurement:
        ramadr  result_HB0              ; Get measurement value for compensated
HB0 result
        move    y, r                    ; Move value to accu. x
        ramadr  init_offset_value       ; This location holds the value of the initial
offset
        move    x, r
        sub     x, y                    ; Subtract the initial offset from the current
measurement value -> result in x

;--- Step Filter ---

        move    z, x                    ; Copy measurement value from x to z
        ramadr  roll_avg_result         ; Get last result (stored in roll_avg_result)
        sub     z, r
        abs     z
        compare z, 20                   ;Check if difference is <> 20
        skipPos 1                       ; If difference is greater than 20, then initial-
ize rolling average new (fast stepping)
```

```
        jsub    roll_avg_initialize

;---- New value in rolling average buffer ----
        jsub    five_times_roll_avg     ;Send new value through 5-time avg filter-> Result will be in x


        move    z, 0x20c114             ; he measured raw value was 4575 with 1kg load. Because of this
; the result has to be multiplied by 0,2186, which is 0.2186 x 2^23

        mult24  x, z                    ; final result in accu. x for no2lcd function

;---- Was tare button pressed ? ----
        ramadr  status_flags            ;Check status register
        skipBitC r, ON_button_rise_flag, 2 ; Was tare button pressed ?
        ramadr  tara_value              ;Store tara value
        move    r, x

;---- Subtract tara value ----
        move    y, x                    ; Store value in y

        ramadr  tara_value
        move    x, r                    ; Put tara value in x
        sub     x, y                    ; Subtract tara value from measured value -> Result in x

;--- Hysteresis for displayed values ----
        move    z, x                    ; Copy result from x to z
        ramadr  last_display_value      ; Ramadr of last displayed value
        gotoEQ  display                 ;value is 0, skip the next lines
        sub     z, r
        abs     z
        ;ramadr state_machine
        ;gotoBitC r, 23, display

        compare z, 6                    ; If the difference is less than 6, don't refresh displayed value
        gotoPos end
```

```
display:
        ramadr      last_display_value      ;Save value which is going to be displayed
        move        r, x

        ramadr      lcd_reg_high
        and         r, 0x000080             ;Erase all special characters which were set by setLCD except 'gr'

        ramadr      state_machine
        bitset      r, 23                   ;Set flag to indicate that displaying on the LCD was done

        no2lcd      x, 1                    ; Function takes x as the value to display and the number of digits after the comma

        ramadr      lcd_reg_mid
        skipPos     1                       ; If value to display is positive skip next two commands
        bitset      r, 22                   ; ... set the minus sign

        newlcd                  ; Show value on LCD display
        goto        end
;-------------------------------------------------------------------------------------------------------------------------
; Add value in x to Rolling Average filter and return averaged value in x again
; (all values are raw values (HB0) and are scaled to gramm later in 'measurement-block')
;-------------------------------------------------------------------------------------------------------------------------
roll_avg_initialize:
        initAvg     x, 5                    ;initialize a 5 times rolling average with the value of x
        ramadr      roll_avg_result
        move        r, x                    ;save result in ramcell roll_avg_result  (used later for five_times_roll_avg subroutine)

        jsubret
```

```
five_times_roll_avg:
        rollAvg     x, 5            ;add value of x to rolling average 5  (which
was previously initialized by initAvg x, 5)
        ramadr      roll_avg_result
        move        r, x            ;save result in ramcell roll_avg_result
(needed for step-filter, hysteresis, etc.)
        jsubret                     ; Return to location from which filter was
called


end:    clrwdt
        stop

;--EOF—
```

AD-MUUNNOSOHJELMA MSP430-MIKROKONTROLLERILLA

```
// AD-muunnos
// Tekijä: Ville Pyykölä

#include "msp430fr5739.h"

void TakeAdcMeas(void);

#define MCLK        24000000


void main( void )
{

  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;



CSCTL0_H = 0xA5;                                        // Unlock register
CSCTL1 |= DCOFSEL0 + DCOFSEL1;                          // Set max. DCO setting
CSCTL2 = SELA__VLOCLK + SELS__DCOCLK + SELM__DCOCLK;
CSCTL3 = DIVA_0 + DIVS_0 + DIVM_0;                      // set all dividers
CSCTL0_H = 0x01;                                        // Lock Register


P1DIR |= BIT0;

// SPI – väylän asetukset (SPI – väylä ei käytössä tässä ohjelmassa)
 // P2SEL1 |= (BIT0 + BIT1);
 // P2SEL0 &= (~(BIT0 + BIT1));

 // P1SEL1 |= (BIT5);
 // P1SEL0 &= ~BIT5;

// Resetistä lähtee toimimaan

  UCA0CTLW0 |= UCSWRST;                     // **Put state machine in reset**
  UCA0CTLW0 |= UCMST+UCSYNC+UCCKPL+UCMSB;   // 3-pin, 8-bit SPI master
                                            // Clock polarity high, MSB

  UCA0CTLW0 |= UCSSEL__SMCLK;                   // ACLK
  UCA0BR0 = 0x02;                           // /2
  UCA0BR1 = 0;                              //
  UCA0MCTLW = 0;                            // No modulation
  UCA0CTLW0 &= ~UCSWRST;                    // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

  // AD – muuntimen asetukset. Jatkuvaa AD – muuntamista yhdeltä kanavalta
  P1SEL1 |= BIT4;
  P1SEL0 &= ~(BIT4);

  // 16ADCclks, MSC, ADC ON
  ADC10CTL0 &= ~ADC10ENC;                       // Ensure ENC is clear
```

```
  ADC10CTL0 = ADC10ON + ADC10SHT_0;
  ADC10CTL1 = ADC10SHS_0 + ADC10SHP + ADC10CONSEQ_0 + ADC10SSEL_0;
  ADC10CTL2 = ADC10RES;
  ADC10CTL2 &= ~(ADC10SR);
  ADC10MCTL0 = ADC10SREF_0 + ADC10INCH_4;
  //ADC10IV = 0x00;                         // Clear all ADC12 channel int flags
  ADC10IE |= ADC10IE0;

  while(1)
  {


    TakeAdcMeas();


  }
}

void TakeAdcMeas(void)
{

  ADC10CTL0 |= ADC10ENC | ADC10SC;          // käynnistetään muunnos
  while ((ADC10IFG & ADC10IFG0)==0);        // onko muunnos valmis?
  P1OUT ^= BIT0;                            // Pinni 1 vaihtaa tilaa
}
```

AD-MUUNNOS SPI-VÄYLÄÄN MSP430-MIKROKONTROLLERILLA

```
// AD-muunnoksen lähetys SPI-väylään
// Tekijä: Ville Pyykölä
//
// AD-muuntimen sisääntulo pinnistä 1.4
// SPI-väylä ulos pinnistä 2.0
//

#include "msp430fr5739.h"


unsigned char ADCResult;
unsigned char RXData =0;
unsigned char TXData;

void main(void)
{
   volatile unsigned int i;

   WDTCTL = WDTPW+WDTHOLD;                    // Stop watchdog timer
   P1DIR |= BIT0;
   P1OUT &= ~BIT0;

   PJDIR |= BIT0 + BIT1 + BIT2;

   PJSEL0 |= BIT4+BIT5;

  // DCO – asetukset

   CSCTL0_H = 0xA5;
   CSCTL1 |= DCOFSEL0 + DCOFSEL1;             // Set max. DCO setting
   CSCTL2 = SELA_0 + SELS_3 + SELM_3;         // set ACLK = XT1; MCLK = DCO
   CSCTL3 = DIVA_0 + DIVS_3 + DIVM_3;         // set all dividers
   CSCTL4 |= XT1DRIVE_0;
   CSCTL4 &= ~XT1OFF;

  // Pinnit

   P1SEL1 |= BIT5;
   P2SEL1 |= BIT0 + BIT1;
   P1SEL1 |= BIT0;

  // SPI - Väylän alustus
  // Resetistä lähtee

   UCA0CTLW0 |= UCSWRST;                      // **Put state machine in reset**
   UCA0CTLW0 |= UCMST+UCSYNC+UCCKPL+UCMSB;    // 3-pin, 8-bit SPI master
                                              // Clock polarity high, MSB

   UCA0CTLW0 |= UCSSEL__SMCLK;                // ACLK
   UCA0BR0 = 0x02;                            // /2
   UCA0BR1 = 0;
   UCA0MCTLW = 0;                             // No modulation
   UCA0CTLW0 &= ~UCSWRST;                     // **Initialize USCI state machine**
   UCA0IE |= UCRXIE;                          // Enable USCI_A0 RX interrupt
```

```
// AD - muuntimen asetukset
// Sisääntulopinnit

  P1SEL1 |= BIT4;
  P1SEL0 |= BIT4;

  ADC10CTL0 &= ~ADC10ENC;                          // Ensure ENC is clear
  ADC10CTL0 = ADC10ON + ADC10SHT_2;
  ADC10CTL1 = ADC10SHS_0 + ADC10SHP + ADC10CONSEQ_2 + ADC10SSEL_0;
  ADC10CTL2 = ADC10RES;
  ADC10CTL2 &= ~ADC10SR;
  ADC10MCTL0 = ADC10SREF_0 + ADC10INCH_4;
  ADC10IV = 0x00;
  ADC10IE |= ADC10IE0;

 // DMA – asetukset
 // ADC10IFG trigger -> AD - muunnokset menevät DMA:n avulla talteen

  DMACTL0 = DMA0TSEL__ADC10IFG;
  __data16_write_addr((unsigned short) &DMA0SA,(unsigned long) &ADC10MEM0);
  // Source single address
  __data16_write_addr((unsigned short) &DMA0DA,(unsigned long) &ADCResult);
  // Destination array address

  DMA0SZ = 64;
  DMA0CTL = DMADT_4 + DMADSTINCR_3 + DMAEN + DMAIE + DMALEVEL;


 while(1)
 {
    UCA0IE |= UCTXIE;
    //__delay_cycles(25000);

    ADC10CTL0 |= ADC10ENC | ADC10SC;
    while ((ADC10IFG & ADC10IFG0)==0);
    ADCResult = ADC10MEM0;                          // Muunnos talteen
    TXData = ADCResult;                             // Lähetettävä data = muunnos

    while (!(UCA0IFG&UCTXIFG));                     // USCI_A0 TX buffer valmis?
    UCA0TXBUF = TXData;                                     // Lähetetään data
    UCA0IE &= ~UCTXIE;


 }
}
```

MSP430-PÄÄOHJELMA

```
/*
 * main.c - DataLogger_prj.

Author: Ville Pyykölä
*/
#include "msp430fr5739.h"
#include "adc.h"                    Author: Texas Instruments
#include "wlan.h"                   Author: Texas Instruments
#include "evnt_handler.h"           Author: Texas Instruments
#include "nvmem.h"                  Author: Texas Instruments
#include "socket.h"                 Author: Texas Instruments
#include "CC3000_common.h"          Author: Texas Instruments
#include "netapp.h"                 Author: Texas Instruments
#include "version.h"                Author: Texas Instruments
#include <string.h>                 Author: Texas Instruments
#include <msp430.h>                 Author: Texas Instruments

#ifndef true
#define true  1
#define false 0
#endif
//
// Simple Config Prefix
//
char aucCC3000_prefix[] = {'P', 'D', 'E'};
//
// These are the main statemachine states
//
#define SEND_THERMOSTAT_STATE        2
#define GET_MESSAGES_FROM_HOST_STATE  3
#define SEND_ACCEL_XYZ_START_STATE    4
#define SEND_ACCEL_XYZ_STATE         5
#define SEND_ACCEL_XYZ_STOP_STATE     6
#define IDLE_STATE                   7

static unsigned char CC3000ConectionState;
//
```

```
// These are the CC3000 connection status states
//
#define CC3000_INIT_STATE            1
#define CC3000_NOT_CONNECTED_STATE    2
#define CC3000_CONNECTED_STATE       3
#define CC3000_CONNECTED_PORTS_OPEN_STATE 4

//
// Various different CC3000 Status values
// returned when attempting to initialize
// and open sockets.
//
typedef enum CC3000_STATUS_T{
  CC3000_CONNECTED_PORTS_OPEN        = 0,
  CC3000_CONNECTED_NO_ADDR_ERR       = -1,
  CC3000_NO_AP_TIMEDOUT_ERR          = -2,
  CC3000_NO_AP_ERR                   = -3,
  CC3000_NO_PORTS_CLOSED_ERR         = -4
}CC3000Status_t;

//
// The Processor Clock frequency in Hz
//
#define MCLK        24000000                              // 24 MHz kellotaajuus
// uncomment this if you would like to use DHCP
//
//#define DHCP_EN                1
//
// Other Wise, you can specify the IP address here
//
unsigned char g_ThisIpAddress[4] = {192, 168, 1, 103};
unsigned char g_ThisMulticastAddress[4] = {224, 0, 0, 1};


//
// Comment out if you want to force a connection to a specific AP
//
//#define SMART_CONFIG_EN        1
// #define TEMPERATURE_UDP_PORT    0xC748
```

```
#define ACCELEROMETER_UDP_PORT  0xC749

//
//  MCLK / 1000 = 1ms tick.
// Do a check that the number will fit in a 16-bit reg
//

#define TICK_RATE_1MS       MCLK / 250000

#if (TICK_RATE_1MS > 0x0000FFFF)
#error "TICK_RATE_1MS > than 16-bit"
#endif

// S1 Mapping (port4)
#define S1_BIT      BIT0
#define S1_PRESSED  ((P4IN & BIT0) ? 0:1)

// LED mapping
#define   LED_CONNECTION      0
#define   LED_TRYING_CONNECT  1
#define   LED_SIMPLE_CONFIG   2
#define   LED_TX              3
#define   LED_RX              4
#define   LED_READY           5
//#define   LED_UNDEF           6
//#define   LED_UNDEF           7

// FRAM Data logger Mem Map
#define SHORT_TERM_BUFFER_SIZE    60
#define LONG_TERM_BUFFER_SIZE     1440
// We don't use the linker here to define the
// the data log memory in FRAM but we do use the
// linker to ensure that we reserve enough
// space.  The linker section is called "FRAM_DATA_LOG"
// Define the end to be even otherwise byte packing wont align
// CSTACK starts at 0xFE00 so we subtract to just to be safe.
// Each buffer has 2 16-bit headers reserved for the Length and the
// index of the T=0 data
#define LONG_TERM_END_ADD         (0xF0A0)
```

```
// We find the start address based off the end address
#define LONG_TERM_START_ADD (LONG_TERM_END_ADD - LONG_TERM_BUFFER_SIZE)
// We reserve two more 16-bit values to store the Length and T=0 position
// This information is sent in the WiFi packet as a header
#define LONG_TERM_HEADER_ADD (LONG_TERM_START_ADD - (2*sizeof(short)))

// Now we repeat the definitions for the short term buffer.
#define SHORT_TERM_END_ADD (LONG_TERM_HEADER_ADD)
#define SHORT_TERM_START_ADD (SHORT_TERM_END_ADD - SHORT_TERM_BUFFER_SIZE)
#define SHORT_TERM_HEADER_ADD (SHORT_TERM_START_ADD - (2*sizeof(short)))

// Stores the pointers to the above buffers
#define FRAM_PTR_SHORT_TERM_LOC (SHORT_TERM_HEADER_ADD - sizeof(short))
#define FRAM_PTR_LONG_TERM_LOC (FRAM_PTR_SHORT_TERM_LOC - sizeof(short))

extern AdcMeasurement_t AdcDevices;

static unsigned short *g_PtrShortTermBuffer, *g_PtrLongTermBuffer;
//static unsigned char g_TimeToTxFlag = true;
static unsigned long g_ulSocket;
static unsigned long g_ulSmartConfigFinished;
static unsigned char g_ucHaveIpAddress = false;
static unsigned char g_DeletePolicy = false;

//Delay100ms
// Performs a long delay of 100ms based on MCLK
void Delay100ms()
{
  __delay_cycles(MCLK/100);
  __no_operation();
}

// SystemInit
void SystemInit(void)
{
        // Startup clock system in max. DCO setting ~8MHz
        // This value is closer to 10MHz on untrimmed parts
```

```
CSCTL0_H = 0xA5;
CSCTL1 |= DCOFSEL0 + DCOFSEL1 + DCORSEL;   // Set max. DCO setting
CSCTL2 = SELA_3 + SELS_3 + SELM_3;
CSCTL3 = DIVA_0 + DIVS_0 + DIVM_0;        // set all dividers

// Turn off temp.
REFCTL0 |= REFTCOFF;
REFCTL0 &= ~REFON;

// Enable switches
// (S1)P4.0 is configured as a switch
// (S2)P4.1  is reserved for the CC3000
P4OUT |= S1_BIT;                  // Configure pullup resistor
P4DIR &= ~(S1_BIT);                // Direction = input
P4REN |= S1_BIT;                  // Enable pullup resistor

// P4.1 - WLAN enable full DS
P4OUT &= ~BIT1;
P4DIR |= BIT1;
P4SEL1 &= ~BIT1;
P4SEL0 &= ~BIT1;

// Enable LEDs
P3OUT &= ~(BIT6+BIT7+BIT5+BIT4);
P3DIR |= BIT6+BIT7+BIT5+BIT4;
PJOUT &= ~(BIT0+BIT1+BIT2+BIT3);
PJDIR |= BIT0 +BIT1+BIT2+BIT3;

// Terminate Unused GPIOs
// P1.0 - P1.6 is unused
P1OUT &= ~(BIT0 + BIT1 + BIT2 + BIT3 + BIT6 + BIT7);
P1DIR &= ~(BIT0 + BIT1 + BIT2 + BIT3 + BIT6 + BIT7);
P1REN |= (BIT0 + BIT1 + BIT2 + BIT3 + BIT6 + BIT7);

// P1.4 is used as input from NTC voltage divider
// Set it to output low
P1OUT &= ~BIT4;
P1DIR |= BIT4;
```

```
// Configure the SPI CS to be on P1.3
P1OUT |= BIT3;
P1DIR |= BIT3;
P1SEL1 &= ~BIT3;
P1SEL0 &= ~BIT3;

// P2.2 - P2.6 is unused
P2OUT &= ~(BIT2 + BIT4 + BIT5 + BIT6);
P2DIR &= ~(BIT2 + BIT4 + BIT5 + BIT6);
P2REN |= (BIT2 + BIT4 + BIT5 + BIT6);

// Configure SPI IRQ line on P2.3
P2DIR  &= (~BIT3);
P2SEL1 &= ~BIT3;
P2SEL0 &= ~BIT3;
// Clear Flag
P2IFG  &= ~BIT3;

// Configure detection of the tripping of Magnetic Sensor on P2.2
P2DIR  &= (~BIT2);
P2SEL1 &= ~BIT2;
P2SEL0 &= ~BIT2;
// Clear Flag
P2IFG  &= ~BIT2;

// Configure UART pins P2.0 & P2.1
//P2SEL1 |= BIT0 + BIT1;
//P2SEL0 &= ~(BIT0 + BIT1);

// P2.7 is used to power the voltage divider for the NTC thermistor
P2OUT &= ~BIT7;
P2DIR |= BIT7;

// P3.0,P3.1 and P3.2 are accelerometer inputs
P3OUT &= ~(BIT0 + BIT1 + BIT2);
P3DIR &= ~(BIT0 + BIT1 + BIT2);
P3REN |= BIT0 + BIT1 + BIT2;

// PJ.0,1,2,3 are used as LEDs
```

```
                // crystal pins for XT1 are unused
                PJOUT &= ~(BIT4+BIT5);
                PJDIR &= ~(BIT4+BIT5);
                PJREN |= BIT4 + BIT5;
}

// PeriodicTimerInit
// Sets up TimerA0 for 1ms periodic timer intervals to take
// ADC sequence samples.

void PeriodicTimerInit(void)
{
  TA0CCTL0 = CCIE;                    // TACCR0 interrupt enabled

  TA0CCR0 = 130;
  TA0CTL = TASSEL_2 + MC_2;           // SMCLK, continuous mode
}

// LedOn
// Sets Led
void LedOn(unsigned char led)
{
  if (led <4)
  {
    PJOUT |= 0x01<<led;
  }
  else
  {
    P3OUT |= 0x01<<led;
  }
}
void LedOff(unsigned char led)
{
  if (led <4)
  {
    PJOUT &= ~(0x01<<led);
  }
  else
  {
```

```
    P3OUT &= ~(0x01<<led);
  }
}
void LedToggle(unsigned char led)
{
  if (led <4)
  {
    if(PJOUT & (0x01<<led) )
    {
      PJOUT &= ~(0x01<<led);
    }
    else
    {
      PJOUT |= 0x01<<led;
    }
  }
  else
  {
    if(P3OUT & (0x01<<led) )
    {
      P3OUT &= ~(0x01<<led);
    }
    else
    {
      P3OUT |= 0x01<<led;
    }
  }
}
void SaveAndResetSystem(void)
{

  // Rest CPU by writing to WD with illegal password.
  //WDTCTL = (0xFF00u) + WDTHOLD;
}

// reportAccelXyz
// Sends the Acceleromter XYZ values to the PC
void reportAccelXyz(void)
{
```

```
  sockaddr tSocketAddr;
  unsigned short usTemp;
  unsigned char ucTX_Buffer[7];

    LedOn(LED_TX);
    {
      // Transmit 7 Bytes
      // Byte 1 is header defining the packet type
      ucTX_Buffer[0] = 0x06;
      usTemp = AdcDevices.AccelX;
      ucTX_Buffer[1] = usTemp;
      ucTX_Buffer[2] = usTemp>>8;

      usTemp = AdcDevices.AccelY;
      ucTX_Buffer[3] = usTemp;
      ucTX_Buffer[4] = usTemp>>8;

      usTemp = AdcDevices.AccelZ;
      ucTX_Buffer[5] = usTemp;
      ucTX_Buffer[6] = usTemp>>8;
      // Write to UDP port
      tSocketAddr.sa_family = AF_INET;
      //
      // port and IP address
      //
      tSocketAddr.sa_data[0] = (unsigned char)(ACCELEROMETER_UDP_PORT>>8);
      tSocketAddr.sa_data[1] = (unsigned char)ACCELEROMETER_UDP_PORT;



      memcpy(&tSocketAddr.sa_data[2], g_ThisMulticastAddress, 4);

      sendto(g_ulSocket, ucTX_Buffer, sizeof(ucTX_Buffer), 0, &tSocketAddr,
            sizeof(sockaddr));
    }
    LedOff(LED_TX);
}

// This function: init_spi
```

```
int init_spi(void)
{
		// Select the SPI lines: MISO/MOSI on P1.6,7 CLK on P2.2
		P1SEL1 |= (BIT6 + BIT7);
		P1SEL0 &= (~(BIT6 + BIT7));

		P2SEL1 |= (BIT2);
		P2SEL0 &= ~BIT2;

		UCB0CTLW0 |= UCSWRST;		// **Put state machine in reset**
		UCB0CTLW0 |= (UCMST+UCSYNC+UCMSB);	// 3-pin, 8-bit SPI master
					// Clock polarity high, MSB
		UCB0CTLW0 |= UCSSEL__SMCLK;	// UCSSEL__SMCLK
		UCB0BR0 = 2;			// /8 of SMCLK = 24Mhz (3mhz)
		UCB0BR1 = 0;			//

		UCB0CTLW0 &= ~UCSWRST;		// **Initialize USCI state machine**

  return(ESUCCESS);
}

// sendDriverPatch
char *sendDriverPatch(unsigned long *Length)
{
		*Length = 0;
		return NULL;
}

// sendBootLoaderPatch
char *sendBootLoaderPatch(unsigned long *Length)
{
		*Length = 0;
		return NULL;
}

//sendWLFWPatch
char *sendWLFWPatch(unsigned long *Length)
{
		*Length = 0;
```

```
	return NULL;
}
// ReadWlanInterruptPin
long ReadWlanInterruptPin(void)
{
	return (P2IN & BIT3);
}

// Enable waln IrQ pin
void WlanInterruptEnable()
{
	__bis_SR_register(GIE);
	P2IES |= BIT3;
	P2IE |= BIT3;
}

// Disable waln IrQ pin
void WlanInterruptDisable()
{
	P2IE &= ~BIT3;
}

// WriteWlanPin
void WriteWlanPin( unsigned char val )
{
	if (val)
	{
		P4OUT |= BIT1;
	}
	else
	{
		P4OUT &= ~BIT1;
	}

}

//CC3000_UsynchCallback

void CC3000_UsynchCallback(long lEventType, char * data, unsigned char length)
```

```
{
        if (lEventType == HCI_EVNT_WLAN_ASYNC_SIMPLE_CONFIG_DONE)
        {
                g_ulSmartConfigFinished = 1;
        }

        if (lEventType == HCI_EVNT_WLAN_UNSOL_CONNECT)
        {
                CC3000ConectionState = CC3000_CONNECTED_STATE;
                // Turn on the LED1
                LedOn(LED_CONNECTION);
#ifndef DHCP_EN
    g_ucHaveIpAddress = true;
#endif
        }

        if (lEventType == HCI_EVNT_WLAN_UNSOL_DISCONNECT)
        {
                CC3000ConectionState = CC3000_NOT_CONNECTED_STATE;
                // Turn off the LED1
                LedOff(LED_CONNECTION);

    // Connection to AP unintentionally closed. Reset
    SaveAndResetSystem();
        }

 if(lEventType == HCI_EVNT_WLAN_UNSOL_DHCP)
 {
    //
    // Should have an IP address now. Can send data.
    //
    g_ucHaveIpAddress = true;
    //
    // Get our assigned address from the passed data
    //
    g_ThisIpAddress[0] = data[3];
    g_ThisIpAddress[1] = data[2];
    g_ThisIpAddress[2] = data[1];
    g_ThisIpAddress[3] = data[0];
```

```
  }
}

// initCC3000Driver
// The function initializes a CC3000 device and triggers it to start operation
int initCC3000Driver(void)
{
                    //
    //init all layers
    //
    init_spi();

// WLAN On API Implementation
                    wlan_init( CC3000_UsynchCallback, sendWLFWPatch, sendDriverPatch,
            sendBootLoaderPatch, ReadWlanInterruptPin, WlanInterruptEnable,
            WlanInterruptDisable, WriteWlanPin);

                    // add just over a 1 second delay to ensure that the slow clock is stable
                    __delay_cycles(2410000);

                    CC3000ConectionState = CC3000_NOT_CONNECTED_STATE;
                    // Trigger a WLAN device.  Set true because there are patches available
                    wlan_start(false);

                    // Mask out all non-required events from CC3000
#ifdef DHCP_EN
    //
    // Setup DHCP.  Although this should be unnecessary as DHCP is the default
    //
    unsigned char pucSubnetMask[4], pucIP_Addr[4], pucIP_DefaultGWAddr[4],
            pucDNS[4];
    memset(pucSubnetMask,       0, sizeof(pucSubnetMask));
    memset(pucIP_Addr,          0, sizeof(pucIP_Addr));
    memset(pucIP_DefaultGWAddr, 0, sizeof(pucIP_DefaultGWAddr));
    memset(pucDNS,              0, sizeof(pucDNS));

    netapp_dhcp((unsigned long *) pucIP_Addr, (unsigned long *) pucSubnetMask,
            (unsigned long *) pucIP_DefaultGWAddr, (unsigned long *) pucDNS);
    wlan_set_event_mask(HCI_EVNT_WLAN_KEEPALIVE |
```

```
                HCI_EVNT_WLAN_UNSOL_INIT |
                HCI_EVNT_WLAN_ASYNC_PING_REPORT);
#else

   // Setup DHCP
                unsigned char pucSubnetMask[4], pucIP_Addr[4], pucIP_DefaultGWAddr[4],
                                                pucDNS[4];
   memset(pucSubnetMask,     0xFF, sizeof(pucSubnetMask));
   memcpy(pucIP_Addr, g_ThisIpAddress, sizeof(g_ThisIpAddress));

   pucIP_DefaultGWAddr[0] = g_ThisIpAddress[0];
   pucIP_DefaultGWAddr[1] = g_ThisIpAddress[1];
   pucIP_DefaultGWAddr[2] = g_ThisIpAddress[2];
   pucIP_DefaultGWAddr[3] = 1;

   memset(pucDNS,            0, sizeof(pucDNS));
   netapp_dhcp((unsigned long *) pucIP_Addr, (unsigned long *) pucSubnetMask,
      (unsigned long *) pucIP_DefaultGWAddr, (unsigned long *) pucDNS);
   wlan_set_event_mask(HCI_EVNT_WLAN_KEEPALIVE  |
                HCI_EVNT_WLAN_UNSOL_INIT |
                HCI_EVNT_WLAN_UNSOL_DHCP |
                HCI_EVNT_WLAN_ASYNC_PING_REPORT);
#endif
  return(0);
}

//FramDataLogInit
void FramDataLogInit(void)
{
  // Get the Ptrs to Short and long Term Buffers
  g_PtrShortTermBuffer = (unsigned short*)FRAM_PTR_SHORT_TERM_LOC;
  g_PtrLongTermBuffer =  (unsigned short*)FRAM_PTR_LONG_TERM_LOC;

  // Check if the pointers are valid.  If not then we are probably runnging for
  // the first time.
  if( (*g_PtrShortTermBuffer > SHORT_TERM_END_ADD) ||
      (*g_PtrShortTermBuffer < SHORT_TERM_START_ADD) ||
      (*g_PtrLongTermBuffer  > LONG_TERM_END_ADD) ||
      (*g_PtrLongTermBuffer  < LONG_TERM_START_ADD) )
```

```
 {
   *g_PtrShortTermBuffer = SHORT_TERM_START_ADD;
   *g_PtrLongTermBuffer =  LONG_TERM_START_ADD;
 }
}

// FramDataLog
void FramDataLog(unsigned char Temperature)
{
 static unsigned char  uc1SecondTicks = 0;
 unsigned char ucOldestShortTermData = 0;

 {
  // First fetch the oldest data that will be over written
  ucOldestShortTermData = (*(unsigned char *)(*g_PtrShortTermBuffer));
  // then Log Data in short Term buffer
  (*(unsigned char *)(*g_PtrShortTermBuffer)) = Temperature;
  // Move Pointer to next available spot
  (*g_PtrShortTermBuffer)++;
  if(*g_PtrShortTermBuffer >= SHORT_TERM_END_ADD)
  {
    (*g_PtrShortTermBuffer) = SHORT_TERM_START_ADD;
  }

  uc1SecondTicks++;
  // 1/minute
  if(uc1SecondTicks >= 60)
  {
    // Log Data in Long Term buffer
    (*(unsigned char *)(*g_PtrLongTermBuffer)) =  ucOldestShortTermData;

    // Move Pointer to next available spot
    (*g_PtrLongTermBuffer)++;
    if(*g_PtrLongTermBuffer >= LONG_TERM_END_ADD)
    {
      (*g_PtrLongTermBuffer) = LONG_TERM_START_ADD;
    }
    uc1SecondTicks = 0;
  }
```

```
  }
}
// StartSmartConfig
//  The function triggers a smart configuration process on CC3000.
//  it exists upon completion of the process
void StartSmartConfig(void)
{
    g_ulSmartConfigFinished = 0;
    // Reset all the previous configuration
    wlan_ioctl_set_connection_policy(0,0,0);
    // Trigger the Smart Config process
    // Start blinking LED2 during Smart Configuration process
    LedOff(LED_SIMPLE_CONFIG);

    wlan_first_time_config_set_prefix(aucCC3000_prefix);
    //
    // Start the Smart Config process
    //
    wlan_first_time_config_start();
    //
    // Wait for simple config finished
    // Set a 1 minute timeout to be safe
    //
    unsigned char ucTimeOut = 60;
    while (g_ulSmartConfigFinished == 0 &&
           ucTimeOut)
    {
      __delay_cycles(6000000);
      LedOn(LED_SIMPLE_CONFIG);
      __delay_cycles(6000000);
      LedOff(LED_SIMPLE_CONFIG);
      ucTimeOut--;
    }

    LedOff(LED_SIMPLE_CONFIG);
    // Configure to connect automatically to the AP retrieved in the
    // simple config process
    wlan_ioctl_set_connection_policy(0, 0, 1);
    // reset the CC3000
```

```
    wlan_stop();
    Delay100ms();
    wlan_start(false);
    // Mask out all non-required events
#ifdef DHCP_EN
    wlan_set_event_mask(HCI_EVNT_WLAN_KEEPALIVE  |
                HCI_EVNT_WLAN_UNSOL_INIT |
                HCI_EVNT_WLAN_ASYNC_PING_REPORT);
#else
    wlan_set_event_mask(HCI_EVNT_WLAN_KEEPALIVE  |
                    HCI_EVNT_WLAN_UNSOL_INIT |
                    HCI_EVNT_WLAN_UNSOL_DHCP |
                    HCI_EVNT_WLAN_ASYNC_PING_REPORT);
#endif
}

// CC3000EnableAndOpenPort
// This function initializes the CC3000 and establishes
// a connection to an AP, then opens a port.  The status of
CC3000Status_t CC3000EnableAndOpenPort( void )
{
  CC3000Status_t Status = CC3000_NO_AP_ERR;
  unsigned char i;
  //
  //  Start the CC3000
  //
  initCC3000Driver();
  //
  // If button is pressed then the user wants to delete the policy
  // so they can connect to a different AP most likely.
  //
  if(g_DeletePolicy == true)
  {
    //
    // Reset all the previous configuration
    //
    wlan_ioctl_set_connection_policy(0,0,0);
    wlan_ioctl_del_profile(0);
    wlan_ioctl_del_profile(1);
```

```
    wlan_ioctl_del_profile(2);
    //
    // reset the CC3000
    //
    wlan_stop();
    initCC3000Driver();

    g_DeletePolicy = false;
  }
  else
  {
    //
    // Configure to connect automatically to the AP retrieved in the
    // simple config process
    //
    wlan_ioctl_set_connection_policy(0, 0, 0);
  }

#ifdef SMART_CONFIG_EN
    //
    // Wait 12 seconds to see if we connect
    //
    for(i = 0; (i < 60) &&
        (CC3000ConectionState == CC3000_NOT_CONNECTED_STATE); i++)
    {
      Delay100ms();
      LedOn(LED_TRYING_CONNECT);
      Delay100ms();
      LedOff(LED_TRYING_CONNECT);
    }
#endif
  //
  // test connection status
  // if we didn't get a connection to an AP then we can try the
  // smart first time config which listens for the secret.
  if(CC3000ConectionState == CC3000_NOT_CONNECTED_STATE)
  {
    //
    // Try Connect to the AP using smart config
```

```
  //
#ifdef SMART_CONFIG_EN
#if (defined(SECURITY_WEP) || defined(SECURITY_WPA) || defined(SECURITY_WPA2))
#error "Error: Smart Config cannot be built with wlan_connect method.  Please Choose only one method"
#endif
  StartSmartConfig();
#else
      // or, direct connect
#if   defined(SECURITY_WEP)
            wlan_connect(1, "DD-WRT", 0x06,NULL, "password", sizeof("password"));
#elif  defined(SECURITY_WPA)
            wlan_connect(2, "DD-WRT", 0x06,NULL, "password", sizeof("password"));
#elif  defined(SECURITY_WPA2)
            wlan_connect(3, "DD-WRT", 0x06,NULL, "password", sizeof("password"));
#else // no security
            wlan_connect(0, "DD-WRT", 0x06, NULL, NULL, 0);
#endif
#endif
  //
  // Wait 3 seconds to see if we connect this time
  //
  for(i = 0; (i < 15) &&
    (CC3000ConectionState == CC3000_NOT_CONNECTED_STATE); i++)
  {
    Delay100ms();
    LedOn(LED_TRYING_CONNECT);
    Delay100ms();
    LedOff(LED_TRYING_CONNECT);
  }
 }
 // Now test we made a connection to the AP
 //
 if(CC3000ConectionState == CC3000_CONNECTED_STATE)
 {
  // wait for an address
#ifdef DHCP_EN
  // wait for a DHCP addr (60 * 0.2s = 12seconds max wait)
  for(i = 0; (i < 60) &&
```

```
    (g_ucHaveIpAddress == false); i++)
   {
     Delay100ms();
     LedOn(LED_TRYING_CONNECT);
     Delay100ms();
     LedOff(LED_TRYING_CONNECT);
   }
#endif

   if( g_ucHaveIpAddress == true)
   {
     //
     // If interested you could get the SSID and ip address now
     // using netapp_ipconfig
     /*static tNetappIpconfigRetArgs ipconfig;
     netapp_ipconfig( &ipconfig);*/
     //
     // Also, If DHCP is used then we also need to verify that we have a valid
     // address. If so then we can now Open UDP socket.
     //
     g_ulSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
     if((signed long)g_ulSocket < 0)
     {
       Status = CC3000_NO_PORTS_CLOSED_ERR;
     }
     else
     {
       Status = CC3000_CONNECTED_PORTS_OPEN;
     }
   }
   else
   {
     Status = CC3000_CONNECTED_NO_ADDR_ERR;
   }
  }
  else
  {
   // No connection to AP. return error
   Status = CC3000_NO_AP_TIMEDOUT_ERR;
```

```
  }

  // Now check the connection status before we leave.  If we were not succesful
  // we should just power down the AP to conserve power unless we made the
  // connection to the AP but do not have a valid Address yet.  If this is
  // the case we don't power down as we want to keep the CC3000 alive long
  // enough to get the DHCP address.
  if( Status != CC3000_CONNECTED_PORTS_OPEN)
  {
    //
    // We were unsuccesful. Power down CC3000
    //
    WriteWlanPin( WLAN_DISABLE );
    CC3000ConectionState = CC3000_NOT_CONNECTED_STATE;
    LedOff(LED_CONNECTION);
  }
  return(Status);
}

void main(void)
{
  static unsigned char ucS1Pressed = false;
  //unsigned char i;
  *g_PtrShortTermBuffer = *g_PtrLongTermBuffer = 0;
  g_ulSocket = 0;
  g_ulSmartConfigFinished = 0;
  g_ucHaveIpAddress = false;
  g_DeletePolicy = false;
  //
  //  Stop WDT
  //
#ifdef __CCS__
                WDTCTL = WDTPW + WDTHOLD;
#elif __IAR_SYSTEMS_ICC__
                asm("mov.w   #0x5a80,&0x015C; ");
#endif
  //
  //  Board Initialization start
  //
```

```
 SystemInit();
 AdcInit();
 //init_spi2();
 //
 // If Button is pressed on power up then flag the CC3000
 // processes to delete the policy the first time through.
 //
 if(S1_PRESSED)
 {
   g_DeletePolicy = true;
 }
 // Enable Global interrupts
 __enable_interrupt();
 PeriodicTimerInit();

 unsigned char state = IDLE_STATE;
 CC3000Status_t CC3000Status;
 LedOn(LED_READY);

 CC3000Status = CC3000EnableAndOpenPort(); // Yhteys päälle heti
 Delay100ms();
 //Delay100ms();
 //Delay100ms();
// Delay100ms();

while(1)
 {
   // Variable initialization
   switch(state)
   {

   case SEND_ACCEL_XYZ_START_STATE:
     // When the button is first pressed we start here and enable the CC3000
     // open the port and send the first pkt

   // CC3000Status = CC3000EnableAndOpenPort();
   // Delay100ms();
     if( CC3000_CONNECTED_PORTS_OPEN == CC3000Status )
     {
```

```
    reportAccelXyz();
    state = IDLE_STATE;
   }
   else
   {
    state = IDLE_STATE;
   }
   break;
 case SEND_ACCEL_XYZ_STATE:

   // Port is already open so we can just continue to send data
   if( CC3000_CONNECTED_PORTS_OPEN == CC3000Status )
   {

    reportAccelXyz();

    state = IDLE_STATE;
   }
   else
   {
    state = IDLE_STATE;
   }
   break;
 case SEND_ACCEL_XYZ_STOP_STATE:
   // Jatkuva yhteys -> ei taukoja
 // WriteWlanPin(WLAN_DISABLE);
  //g_ucHaveIpAddress = false;
  //CC3000ConectionState = CC3000_NOT_CONNECTED_STATE;
  // LedOff(LED_CONNECTION);
   state = IDLE_STATE;
   break;
 case GET_MESSAGES_FROM_HOST_STATE:
   // This application does not receive
   state = IDLE_STATE;
   break;
 case IDLE_STATE:
 default:
   if( (CC3000_CONNECTED_STATE == CC3000ConectionState) ||
```

```
    (CC3000_CONNECTED_PORTS_OPEN_STATE == CC3000ConectionState) )
{
  hci_unsolicited_event_handler();
}

//
// Check the state of S1.  S1 is used to
// control the stream of Accelerometer data
// to the PC.  While the Button is asserted
// The Acceleromter data is streamed.
if(S1_PRESSED)
{
  // See if the swwitch was just pressed.
  // If so, then we need to first enable the CC3000
  // and send the first packet.

  if(ucS1Pressed == false)
  {

    ucS1Pressed = true;
    state = SEND_ACCEL_XYZ_START_STATE;
  }
  else
  {
    // Continue to send.  CC3000 is already powered up

    state = SEND_ACCEL_XYZ_STATE;
  }
}
else
{
  // See if the switch was just released
  if(ucS1Pressed == true)
  {
    ucS1Pressed = false;
    // Done sending. Turn off the CC3000 to conserve power.
    state = SEND_ACCEL_XYZ_STOP_STATE;
  }
  else
```

```
      {
        state = IDLE_STATE;
      }
     }
     break;
   }
  }
}

// Interrupt Service Routines
// Timer A0 ISR for MODE2
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{

  TA0CCR0 += 130;
  AdcUpdate();

}
```

MSP430-PÄÄOHJELMAN AD-MUUNNOSOHJELMA

```
// Author: Ville Pyykölä

#include "msp430fr5739.h"
#include "adc.h"
// 8-bit ADC conversion result array
volatile unsigned short ADC_Result[SIZE_OF_SEQUENCE];

AdcMeasurement_t AdcDevices;
unsigned char TXData;

char mtype;
volatile unsigned int XValue = 0;
volatile unsigned int YValue = 0;
volatile unsigned int ZValue = 0;

void AdcInit (void)
{
  //
  // Accelerometer
  // Configure ADC pins. Enable A/D channel inputs
  //
  ACC_PORT_SEL0 |= ACC_X_PIN + ACC_Y_PIN + ACC_Z_PIN;
  ACC_PORT_SEL1 |= ACC_X_PIN + ACC_Y_PIN + ACC_Z_PIN;
  ACC_PORT_DIR &= ~(ACC_X_PIN + ACC_Y_PIN + ACC_Z_PIN);

  // Enable ACC_POWER.  Actually enables thermistor power too.
 ACC_PWR_PORT_DIR |= ACC_PWR_PIN;
 ACC_PWR_PORT_OUT |= ACC_PWR_PIN;

  // Configure ADC
  //Enable A/D channel inputs
  THERM_PORT_SEL0 |= THERM_IN_PIN;
  THERM_PORT_SEL1 |= THERM_IN_PIN;
  THERM_PORT_DIR &= ~(THERM_IN_PIN);

  ADC10CTL0 &= ~ADC10ENC;                    // Ensure ENC is clear
  ADC10CTL0 = ADC10ON + ADC10SHT_2 + ADC10MSC;
```

```
  ADC10CTL1 = ADC10SHS_0 + ADC10SHP + ADC10CONSEQ_1 + ADC10SSEL_0;
  ADC10CTL2 = ADC10RES;
  ADC10MCTL0 = ADC10SREF_0 + ADC10INCH_14;



  // Configure DMA0 (ADC10IFG trigger)
  // ADC10IFG trigger
  DMACTL0 = DMA0TSEL__ADC10IFG;
  __data16_write_addr((unsigned short) &DMA0SA,(unsigned long) &ADC10MEM0);
  // Source single address
  __data16_write_addr((unsigned short) &DMA0DA,(unsigned long) &ADC_Result[0]);
  // Destination array address
  // 15 conversions
  DMA0SZ = SIZE_OF_SEQUENCE;
  DMA0CTL = DMADT_4 + DMADSTINCR_3 + DMAEN + DMAIE;
}

// TakeADCMeas
// Take ADC measurement

void TakeAdcMeas(void)
{
  //while (ADC10CTL1 & BUSY);

  ADC10CTL0 |= ADC10ENC | ADC10SC ;
  P1OUT ^= BIT0;
  // __bis_SR_register(GIE);
  // __no_operation();                    // For debug only
}

//  AdcUpdate
// Takes a Temperature and Accel XYZ Measurement and stores them.
void AdcUpdate(void)
{
  TakeAdcMeas();
  // Update the Temperature



  AdcDevices.AccelX = ADC_Result[ACC_X_IDX];
```

```
  AdcDevices.AccelY = ADC_Result[ACC_Y_IDX];
  AdcDevices.AccelZ = ADC_Result[ACC_Z_IDX];

  /*
   TXData = ADC_Result[0];
   while (!(UCA0IFG&UCTXIFG));              // USCI_A0 TX buffer ready?
    UCA0TXBUF = TXData;                     // Lähetetään data

  */

}

#pragma vector=DMA_VECTOR
__interrupt void DMA0_ISR (void)
{
  switch(__even_in_range(DMAIV,16))
  {
    case  0: break;                         // No interrupt
    case  2:
       // sequence of conversions complete
     // __bic_SR_register_on_exit(CPUOFF);
      break;                                // DMA0IFG
    case  4: break;                         // DMA1IFG
    case  6: break;                         // DMA2IFG
    case  8: break;                         // Reserved
    case 10: break;                         // Reserved
    case 12: break;                         // Reserved
    case 14: break;                         // Reserved
    case 16: break;                         // Reserved
    default: break;
  }

}
```


SAVONIA